A Decision-Theoretic Approach to Reliable Message Delivery

Francis C. Chu Joseph Y. Halpern

Department of Computer Science
Upson Hall, Cornell University
Ithaca, NY 14853-7501, USA

{fcc,halpern}@cs.cornell.edu

To be, or not to be: that is the question:
Whether 'tis nobler in the mind to suffer
The slings and arrows of outrageous fortune,
Or to take arms against a sea of troubles,
And by opposing end them?

Hamlet (III, i)

Abstract

We argue that the tools of decision theory should be taken more seriously in the specification and analysis of systems. We illustrate this by considering a simple problem involving reliable communication, showing how considerations of utility and probability can be used to decide when it is worth sending heartbeat messages and, if they are sent, how often they should be sent.

Keywords: decision theory, specifications, design and analysis of distributed systems

1 Introduction

In designing and implementing systems, choices must always be made: When should we garbage collect? Which transactions should be aborted (to remove a deadlock)? How big should the page table be? How often should we resend a message that is not acknowledged? Currently, these decisions seem to be made based on intuition and experience. However, studies suggest that decisions made in this way are prone to inconsistencies and other pitfalls [RS89]. Just as we would like to formally verify critical programs in order to avoid bugs, we would like to apply formal methods when making important decisions in order to avoid making suboptimal decisions. Mathematical logic has given us the tools to verify programs, among other things. There are also standard mathematical tools for making decisions, which come from decision theory [Res87]. We believe that these tools need to be taken more seriously in systems design. We view this paper as a first step towards showing how this can be done and the benefits of so doing.

Before we delve into the technical details, let us consider a motivating example. Suppose Alice made an appointment with Bob and the two are supposed to meet at five. Alice shows up at five on the dot but Bob is nowhere in sight. At 5:20, Alice is getting restless. The question is "To stay or not to stay?" The answer, of course, is "It depends." Clearly, if Bob is an important business

This work was supported in part by NSF grant IRI-96-25901 and by the Air Force Office of Scientific Research grant F49620-96-1-0323.

client and they are about to close a deal, she might be willing to wait longer. On the other hand, if Bob is an in-law she never liked, she might be happy to have an excuse to leave. At a more abstract level, the utility of actually having the meeting is (or, at least, should be) an important ingredient in Alice's calculations. But there is another important ingredient: likelihood. If Alice and Bob meet frequently, she may know something about how prompt he is. Does he typically arrive more or less on time (in which case the fact that he is twenty minutes late might indicate that he is unlikely to come at all) or is he someone who quite often shows up half an hour late? Not surprisingly, utilities and probabilities (as measures of likelihood) are the two key ingredients in decision theory.

While this example may seem far removed from computer systems, it can actually be viewed as capturing part of atomic commitment [SKS97]. To see this, suppose there is a coordinator p_c and two other processes p_a and p_b working on a transaction. To commit the transaction, the coordinator must get a yes vote from both p_a and p_b. Suppose the coordinator gets a yes from p_a, but hears nothing from p_b. Should it continue to wait or should it abort the transaction? The types of information we need to make this decision are precisely those considered in the Alice-Bob example above: probabilities and utilities. While it is obvious that the amount of time Alice should wait depends on the situation, atomic commit protocols typically have a context-independent timeout period. If p_c has not heard from all the processes by the end of the timeout period, then the transaction is aborted. Since the importance of the transaction and the cost of waiting are context-dependent, the timeout period would not be appropriate in every case.

Although it is not done in atomic commit protocols, there certainly is an awareness that we need to take utilities or costs into account elsewhere in the database literature.[1] For example, when a deadlock is detected in a database system, some transaction(s) must be rolled back to break the deadlock. How do we decide which ones? The textbook response [SKS97, p. 497] is that “[we] should roll back those transactions that will incur the minimum cost. Unfortunately, the term minimum cost is not a precise one.” Typically, costs have been quantified in this context by considering things like how long the transaction has been running and how much longer it is likely to run, how many data items it has used, and how many transactions will be involved in a rollback. This is precisely the type of analysis to which the tools of decision theory can be applied. Ultimately we are interested in when each transaction of interest will complete its task. However, some transactions may be more important than others. Thus, ideally, we would like to attach a utility to each vector of completion times. Of course, we may be uncertain about the exact outcome (e.g., the exact running time of a transaction). This is one place where likelihood enters the picture. Thus, in general, we will need both probabilities and utilities to decide which are the most appropriate transactions to abort. Of course, obtaining the probabilities and utilities may in practice be difficult. Nevertheless, we may often be able to get reasonable estimates of them (see Section 6 for further discussion of this issue), and use them to guide our actions.

In this paper, we illustrate how decision theory can be used and some of the subtleties that arise in using it. We focus on one simple problem involving reliable communication. For ease of exposition, we make numerous simplifying assumption in our analysis. Despite these simplifying assumptions, we believe our results show that decision theory can be used in the specification and design of systems.

We are not the first to attempt to apply decision theory in computer science. Shenker and his colleagues [BBS98, BS98], for example, have used ideas from decision theory to analyze various

[1]Awareness of cost is by no means limited to the database community. For example, a sampling of the papers at a recent DISC (Distributed Computing) Conference, showed that cost was mentioned in at least seven of them [BMPP98, CM98, EHWG98, FMS98, MIB98, TRAR98, YAGW98]. Cost and utility are also discussed, for example, in [Kes97] and [KL95, LS98].

network protocols; Microsoft has a Decision Theory and Adaptive Systems group that has successfully used decision theory in a number of applications, including troubleshooting problems with printers and intelligent user interfaces in Office '97. (See http://research.microsoft.com/dtas/ for further details.) Mikler et al. [MHW96] have looked at network routing from a utility-theoretic perspective. One important difference between our paper and theirs is that they do not treat the utility function as a given: Their aim is to find a good utility function so that the routing algorithm would exhibit the desired behavior (of avoiding the hot spot). More generally, our focus on writing specifications in terms of utility, and the subtleties involved with the particular application we consider here—reliable communication—make the thrust of this paper quite different from others in the literature.

The rest of this paper is organized as follows. We briefly review some decision-theoretic concepts in Section 2. In Section 3 we describe the basic model and introduce the communication problem that serves as our running example. We show that the expected cost of even a single attempt at reliable communication is infinite if there is uncertainty about process failures. We then show in Section 4 how we can achieve reliable communication with finite expected cost by augmenting our system with heartbeat messages, in the spirit of Aguilera, Chen, and Toueg [ACT97]. However, the heartbeat messages themselves come at a cost; this cost is investigated in Section 5. We offer some conclusions in Section 6. Some proofs are relegated to the appendix.

2 A Brief Decision Theory Primer

The aim of decision theory is to help agents make rational decisions. There are a number of equivalent ways of formalizing the decision process. In this paper, we assume that (a) we have a set O of possible states of the world or outcomes, (b) the agent can assign a utility from $\mathbb{R} \cup \{-\infty, \infty\}$ (denoted $\mathbb{R}^*$) to each outcome in O, and (c) each action or choice a of the agent can be associated with a subset O_a of O and a probability measure $\Pr_a$ on O_a. (This is essentially equivalent to viewing $\Pr_a$ as a probability measure on O which assigns probability 0 to the outcomes in $O - O_a$.)

Roughly speaking, the utility associated with an outcome measures how happy the agent would be if that outcome occurred. Thus, utilities quantify the preferences of the agent. The agent prefers outcome o_1 to outcome o_2 iff the utility of o_1 is higher than that of o_2. The set O_a of outcomes associated with an action or choice a are the outcomes that might arise if a is performed or chosen; the probability measure on O_a represents how likely each outcome is if a is performed. These are highly nontrivial assumptions, particularly the last two. We discuss them (and to what extent they are attainable in practice) in Section 6. For now, though, we just focus on their consequences.

Recall that a random variable on the set O of outcomes is a function from O to $\mathbb{R}^*$. Given a random variable X and a probability measure $\Pr$ on the outcomes, the expected value of X with respect to $\Pr$, denoted $E^{\Pr}(X)$, is $\sum_{v \in X(O)} v \Pr(X = v)$, where $X(O)$ is the range of X and $X = v$ denotes the set $\{o \in O : X(o) = v\}$. We drop the superscript $\Pr$ if it is clear from the context. Note that utility is just a random variable on outcomes. Thus, with each action or choice, we have an associated expected utility, where the expectation is taken with respect to O_a and $\Pr_a$. Since utilities can be infinite, we need some conventions to handle infinities in arithmetic expressions. If $x > 0$, we let $x \times \infty = \infty$; if $x < 0$, we let $x \times \infty = -\infty$. For all $x \in \mathbb{R}$, we let $x + \infty = \infty$. Finally, we let $0 \times \infty = 0$. We assume that $+$ and $\times$ remain commutative on $\mathbb{R}^*$, so this covers all the cases but $\infty + (-\infty)$, which we take to be undefined.

The "rational choice" is typically taken to be the one that maximizes expected utility. While other notions of rationality are clearly possible, for the purposes of this paper, we focus on expected utility maximization. Again, see Section 6 for further discussion of this issue.

We can now apply these notions to the Alice-Bob example from the introduction. One way of characterizing the possible outcomes is as pairs $(m_a; m_b)$, where m_a is the number of minutes that Alice is prepared to wait, and m_b is the time that Bob actually arrives. (If Bob does not arrive at all, we take $m_b = \infty$.) Thus, if $m_a \geq m_b$, then Alice and Bob meet at time m_b in the outcome $(m_a; m_b)$. If $m_a < m_b$, then Alice leaves before Bob arrives. What is the utility of the outcome $(m_a; m_b)$? Alice and Bob may well assign different utilities to these outcomes. Since we are interested in Alice's decision, we consider Alice's utilities. A very simple assumption is that there is a fixed positive benefit meet-Bob to Alice if she actually meets Bob and a cost of c-wait for each minute she waits, and that these utilities are additive. We assume here that c-wait ≤ 0. (In general, costs are described by non-positive utilities.) Under this assumption, the utility of the outcome $(m_a; m_b)$ is meet-Bob + m_bc-wait if $m_a \geq m_b$ and m_ac-wait if $m_a < m_b$.

Of course, in practice, the utilities might be much more complicated and need not be additive. For example, if Alice has a magazine to read, waiting for the first fifteen minutes might be relatively painless, but after that, she might get increasingly frustrated and the cost of waiting might increase exponentially, not linearly. The benefit to meeting Bob may also depend on the time they meet, independent of Alice's frustration. For example, if they have a dinner reservation for 6 p.m. at a restaurant half an hour away, the utility of meeting Bob may drop drastically after 5:30. Finally, the utility of $(m_a; m_b)$ might depend on m_b even if $m_a < m_b$. For example, Alice might feel happier leaving at 5:15 if she knew that Bob would arrive at 6:30 than if she knew he would arrive at 5:16.

Once Alice has decided on a utility function, she has to decide what action to take. The only choice that Alice has is how long to wait. With each choice m_a, the set of possible outcomes consists of those of the form $(m_a; m_b)$, for all possible choices of m_b. Thus, to compute the expected utility of the choice m_a, she needs a probability measure over this set of outcomes, which effectively means a probability measure over Bob's possible arrival times.

This approach of deciding at the beginning how long to wait may seem far removed from actual practice, but suppose instead Alice sent her assistant Cindy to meet Bob. Knowing something about Bob's timeliness (or lack thereof), she may well want to give Cindy instructions for how long to wait. Taking the cost of waiting to be linear in the amount of time that Cindy waits is now not so unreasonable, since while Cindy is tied up waiting for Bob, she is not able to help Alice in other ways. If Cindy goes to meet Bob frequently for Alice, it may make more sense for Alice just to tell Cindy her utility function, and let Cindy decide how long to wait based on the information she acquires regarding Bob's punctuality. Of course, once we think in terms of Alice sending an assistant, it is but a small step to think of Alice running an application, and giving the application instructions to help it decide how to act.

3 Reliable Communication

We now consider a problem that will serve as a running example throughout the rest of the paper. Consider a system consisting of a sender p and a receiver q connected by an unreliable bidirectional link. We assume that the link satisfies the following properties:

- The transmission delay of the link is δ.
- The link can only fail by losing (whole) messages and the probability of a message loss is γ.

We assume that the transmission delay and the probability of message loss are independent of the state of the system.[2] A process is correct if it never crashes. For $x \in \{p, q\}$, let α_x be the probability

[2] The results of this paper hold even if these quantities do depend on the state of the link. For example, γ may be a function of the number of messages in transit. We stick to the simpler model for ease of exposition.

that x is correct (more precisely, the probability of the set of runs in which x is correct). In runs in which x is not correct, x crashes in each time unit with probability $_x > 0$, independent of all other events in the system (such as the events that occurred during the previous time unit).

The assumptions that seems most reasonable to us is that $_p = {}_q = 0$: in practice, there is always a positive probability that a process will crash in any given round.[3] We allow the possibility that $_x \neq 0$ to facilitate comparison to most of the literature, which does not make probabilistic assumptions about failure. It also may be a useful way of modeling the scenario in which processes stay up forever "for all practical purposes" (for example, if the system is scheduled to be taken o -line before the processes crash).

We want to implement a reliable link on top of the unreliable link provided by the system. That is, we want to implement a reliable send-receive protocol SR using the (unreliable) sends and receives provided by the link, denoted send and receive. SR is a joint protocol, consisting of a SEND protocol for the sender and a RECEIVE protocol for the receiver. SR can be initiated by either p or q. A send-receive protocol is said to be sender-driven if it is initiated by p and receiver-driven if it is initiated by q. (Web browsing can be viewed as an instance of a receiver-driven activity. The web browser queries the web server for the content of the page.) We assume that sends and receives take place at a time t, while SENDs and RECEIVEs take place over an interval of time (since, in general, they may involve a sequence of sends and receives).

We assume that send and receive satisfy the following two properties:

- If q receives m at time t, then p sent m at time t and m was not lost (since the link cannot create messages or duplicate messages and the transmission delay is known to be).

- If p sends m at time t, then with probability 1 , q will receive m at time t + ; if q does not receive m at time t + , q will never receive it.

What speci cation should SR satisfy? Clearly we do not want the processes to create messages out of whole cloth. Thus, we certainly want the following requirement:

S_0. If q nishes RECEIVing m at time t, then p must have started SENDing m at some time $t' \;\; t$ and q must have received m at some time $t'' \;\; t$.

We shall implicitly assume S_0 without further comment throughout the paper.

The more interesting question is what liveness requirements SR should satisfy. Perhaps the most obvious requirement is:

S_1. If p and q are correct and SR is started with m as the message, then q eventually nishes RECEIVing m.

Although S_1 is very much in the spirit of typical speci cations, which focus only on what happens if processes are correct, we would argue that it is rather uninteresting, for two reasons (which apply equally well to many other similar speci cations). The rst shows that it is too weak: If $_p = {}_q = 0$, then p and q are correct (i.e., never crash) with probability 0. Thus, speci cation S_1 is rather uninteresting in this case: It is saying something about a set of runs with vanishingly small likelihood. The second problem shows that S_1 is too strong: In runs where p and q are correct, there is a chance (albeit a small one) that the link may lose all messages. In this case, q cannot nish RECEIVing m, since it cannot receive m (as all the messages are lost). Thus S_1 is not satis ed.

[3] We assume that round k takes place between time k 1 and k.

Of course, both of these problems are well known. The standard way to strengthen S_1 to deal with the first problem is to require only that p and q be correct for "sufficiently long", but then we need to quantify this; it is far from clear how to do so. The standard way to deal with the second problem is to restrict attention to fair runs, according to some notion of fairness [Fra86], and require only that q finishes RECEIVing m in fair runs. Fairness is a useful abstraction for helping us characterize conditions necessary to prove certain properties. However, what makes fairness of practical interest is that, under reasonable probabilistic assumptions, it holds with probability 1.

Our interest here, as should be evident from the introduction, is to make more explicit use of probability in writing a specification. For example, we can write a probabilistic specification like the following:

S_2. $\lim_{t \to \infty} \Pr$(q finishes RECEIVing m no later than t time units after the start of SR | p and q are up t time units after the start of SR) = 1.

Requirement S_2 avoids the two problems we saw with S_1. It says, in a precise sense, that if p and q are up for sufficiently long, then q will RECEIVE m with high probability (where "sufficiently long" is quantified probabilistically). Moreover, by making only a probabilistic statement, we do not have to worry about unfair runs: They occur with probability 0.

The traditional approach has been to separate specifying the properties that a protocol must satisfy from the problem of finding the best algorithm that meets the specification. But that approach typically assumes that properties are all-or-nothing propositions. That is, it implicitly assumes that a desirable property must be true in every run (or perhaps every fair run) of a protocol. It does not allow a designer to specify that it may be acceptable for a desirable property to sometimes fail to hold, if that results in much better properties holding in general. We believe that, in general, issues of cost should not be separated from the problem of specifying the behavior of an algorithm. A protocol that satisfies a particular traditional specification may do so at the price of having rather undesirable behavior on a significant fraction of runs. For example, to ensure safety, a protocol may block 20% of the time. There may be an alternate protocol that is unsafe only 2% of the time but also blocks only 2% of the time. Whether it is better to violate safety 2% of the time and liveness 2% of the time or to never violate safety but violate liveness 20% of the time obviously depends on the context. The problem with the traditional approach is that this comparison is never even considered (any algorithm that does not satisfy safety is automatically dismissed).

While we believe S_2 is a better specification of what is desired than S_1, it is still not good enough for our purposes, since it does not take costs into account. Without costs, we still cannot decide if it is better to violate liveness 20% of the time or to violate safety 2% of the time and liveness 2% of the time. As a first step to thinking in terms of costs, consider the following specification:

S_3. For each message m, the expected cost of SR(m) is finite.

As stated, S_3 is not well defined, since we have not specified the cost function. We now consider a particularly simple cost function, much in the spirit of the Alice-Bob example discussed in Section 2. Let SR be a send-receive protocol. Its outcomes are just the possible runs or executions. We want to associate with each run its utility. There are two types of costs we will take into account: sending messages and waiting. The intuition is that each attempt to send a message consumes some system resources and each time unit spent waiting costs the user. The total cost is a weighted sum of the two.

More precisely, let c-send and c-wait be constants representing the cost of sending a message and of waiting one time unit, respectively. Given a run r, let #-send(r) be the number (possibly

∞) of sends done by the protocol in run r. We now want to define t-wait(r), which intuitively is the amount of time q spends waiting to RECEIVE m. When should we start counting? In the Alice-Bob example, it was clear, since Alice starts waiting for Bob at 5:00. We do not want to start counting at a fixed time, since we do not assume that the processes will start their protocol at a particular time. What we want is to start at the time when SR is invoked. When do we stop counting, assuming we started? If there are no process crashes, then we stop counting when q finishes RECEIVing m. What if there are process crashes? In traditional specifications (such as S_1), the protocol has no obligations once a process fails. To facilitate comparison between our approach and the traditional approach, we stop counting at the time of a process crash if it happens before q finishes RECEIVing m. (Note that q may never finish RECEIVing if a process crashes.)

Let t_s be the time SR is invoked. (If no such time exists, we let t-wait$(r) = 0$.) Let t_p be the time p crashes ($t_p = \infty$ if p does not crash); let t_q be the time q crashes ($t_q = \infty$ if q does not crash); let t_f be the time q finishes RECEIVing m ($t_f = \infty$ if q does not finish). Finally let t-wait$(r) = \max\{\min\{t_p; t_q; t_f\}; t_s\} - t_s$. We take the (total) cost of run r to be

$$c_0(r) = \#\text{-send}(r)c\text{-send} + \text{t-wait}(r)c\text{-wait}:$$

Note that c_0 is a random variable on runs. If $c_0(r)$ captures the cost of run r (as we are assuming here it does), then S_3 says that we want $E(c_0) = E(\#\text{-send})c\text{-send} + E(\text{t-wait})c\text{-wait}$ to be finite.

Note that, if SR is not invoked in a run r, then $c_0(r) = 0$. Since we are interested in the expected cost of SR, we consider only runs in which SR is actually invoked. Also, since we are interested in the expected cost of a single invocation in this (and the next) section, we assume for ease of exposition that the protocol is invoked at time 0 (so t-wait$(r) = \min\{t_p; t_q; t_f\}$) throughout these two sections without further comment.

Proposition 3.1: S_2 and S_3 are incomparable under cost function c_0.

Proof: Suppose $\alpha_p = \alpha_q = 1$. Consider a send-receive protocol SR_0 in which p sends m in every round until it receives ack(m), and q sends its kth ack(m) N^k rounds after receiving m for the kth time, where $N\gamma > 1$. (Recall that γ is the probability of message loss.) It is easy to see that SR_0 satisfies S_2. We show that it does not satisfy S_3 by showing that $E(\#\text{-send}) = \infty$.

The basic idea is that q is not acknowledging the receipt of m in a timely fashion, so p will send too many copies of m. Let $A_k = \{r : q\text{'s first } k \text{ acks are lost and the } (k+1)\text{st ack makes it in } r\}$; let $A_\infty = \{r : \text{all of } q\text{'s acks are lost}\}$. Note that $Pr(A_k) = \gamma^k(1-\gamma)$ and $Pr(A_\infty) = 0$ (so we can ignore runs in A_∞ for the purpose of computing expected cost, since we adopted the convention that $0 \cdot \infty = 0$). Note also that $E(\#\text{-send} \mid A_k) \geq N^k$, since p cannot possibly get its first ack(m) before time N^k in runs in A_k. Thus

$$E(\#\text{-send}) = \sum_{k=0}^{\infty} E(\#\text{-send} \mid A_k) Pr(A_k) \geq \sum_{k=0}^{\infty} N^k \gamma^k (1-\gamma):$$

It is clear that the last sum is not finite, since $N\gamma > 1$; thus the algorithm fails to satisfy S_3.

Suppose $\alpha_p = \alpha_q = 0$. Consider the trivial protocol (i.e., the "do nothing" protocol). In a round in which both p and q are up, one of p or q will crash in the next round with probability $\alpha = \alpha_p + \alpha_q - \alpha_p\alpha_q$. So the probability that the first crash happens at time k is $(1-\alpha)^k\alpha$. Thus one of them is expected to crash at time

$$\begin{aligned}\sum_{k=0}^{\infty} k(1-\alpha)^k\alpha &= \frac{\alpha(1-\alpha)}{(1-(1-\alpha))^2}\\ &= \frac{1-\alpha}{\alpha}:\end{aligned}$$

(Here and elsewhere in this paper we use the well-known fact that $\sum_{k=0}^{\infty} kx^k = \frac{x}{(1\ x)^2}$.) Thus, $E(c_0) = \frac{1}{\ }$ c-wait for the trivial protocol, so the trivial protocol satis es S_3, although it clearly does not satisfy S_2. ∎

The following theorem characterizes when S_3 is implementable with respect to the cost function c_0. Moreover, it shows that with this cost function, when S_3 is satis able, there are in fact protocols that satisfy S_3 and S_2 simultaneously.

Theorem 3.2: Under cost function c_0, there is a send-receive protocol satisfying S_3 i $\ _p = 0$ or $\ _q = 0$ or $\ _q = 1$ or $\ _p = 1$. Moreover, if $\ _p = 0$ or $\ _q = 0$ or $\ _q = 1$ or $\ _p = 1$, then there is a send-receive protocol that satis es both S_2 and S_3.

Proof: Suppose $\ _q = 1$ or $\ _p = 0$. Consider the (sender-driven) protocol SR_1 in which p sends m to q until p receives an ack(m) from q, and q sends ack(m) whenever it receives m. SR_1 starts when p rst sends m and q nishes RECEIVing m when it rst receives m. To see that SR_1 is correct, rst consider the case that $\ _q = 1$. Let $C_p = \{r : $ p receives ack(m) at least once from q in $r\}$. Let $N_1(r) = k_1$ if the k_1th copy of m is the rst received by q and let $N_2(r) = k_2$ if the k_2th copy of m is the one whose corresponding ack(m) is the rst received by p.

Since the probability that the link may drop a particular message is ,

$$E(N_1 \mid C_p) = \sum_{k=1}^{\infty} k\ ^{k\ 1}(1\ \ ) = \frac{1\ \ }{\ }\sum_{k=1}^{\infty} k\ ^k = \frac{1\ \ }{\ }\frac{\ }{(1\ \ )^2} = \frac{1}{1\ \ }:$$

An analogous argument shows that $E(N_2 \mid C_p) = \frac{1}{(1\ \ )^2}$. Note that t-wait$(r) = N_1(r) + \ \ 1$ for $r \in C_p$, so $E(\text{t-wait} \mid C_p) = E(N_1 \mid C_p) + (\ \ 1) = \frac{1}{(1\ \ )} + \ \ 1$. Moreover, since p stops sending m when it receives ack(m) from q, it will stop $2\ $ rounds after the $N_2(r)$th send of m in run r. Thus $\frac{1}{(1\ \ )^2} + 2\ \ 1$ is the number of times p is expected to send m in runs of C_p. We expect $1\ $ of these to be successful, so the number of times q is expected to send ack(m) is at most $\frac{1}{(1\ \ )} + (2\ \ 1)(1\ \ )$. (The actual expected value is slightly less since q may crash shortly after sending the rst ack(m) received by p in runs of C_p). We conclude that $E(\#\text{-send} \mid C_p)\ \ \frac{1}{(1\ \ )} + \frac{1}{(1\ \ )^2} + (2\ \ 1)(2\ \ )$. Thus $E(c_0 \mid C_p)$ is nite, since both $E(\#\text{-send} \mid C_p)$ and $E(\text{t-wait} \mid C_p)$ are nite.

We now turn to $E(c_0 \mid \overline{C_p})$. We rst partition $\overline{C_p}$ into two sets:

$F_1 = \{r :$ p crashes before receiving an ack(m) from $q\}$ and

$F_2 = \{r :$ p does not crash and does not receive ack(m) from $q\}$.

Note that $Pr(F_2) = 0$ and $Pr(F_1) = 1\ \ Pr(C_p)$. We may ignore runs of F_2 for the purposes of computing the expected cost since we adopted the convention that $0\ \ 1 = 0$. In runs r of F_1, t-wait(r) is at most the time it takes for p to crash, which is expected to occur at time $\frac{1\ \ _p}{\ _p}$. Thus $E(\text{t-wait} \mid F_1) < \frac{1}{\ _p}$. Furthermore, if p crashes at time t_c in $r \in F_1$, it sends m exactly t_c times in r (since p does not receive ack(m) in runs of F_1). In that case, q sends ack(m) at most t_c times. So $\#\text{-send}(r)\ \ 2t_c$ if p crashes at time t_c in $r \in F_1$. Thus $E(\#\text{-send} \mid F_1) < \frac{2}{\ _p}$. It follows that $E(c_0 \mid \overline{C_p})$ is nite. Since both $E(c_0 \mid C_p)$ and $E(c_0 \mid \overline{C_p})$ are nite, $E(c_0)$ is nite; so SR_1 satis es S_3. To see that the protocol satis es S_2, note that for $t\ \ \ $, the probability that q does not nish RECEIVing m by time t given that both p and q are still up is $\ ^{t\ \ }$. Thus S_2 is also satis ed.

Now consider the case that $\ _p = 0$. Note that in this case, p is expected to crash at time $\frac{1\ \ _p}{\ _p}$. Thus, $E(\text{t-wait}) < \frac{1}{\ _p}$ and $E(\#\text{-send}) < \frac{2}{\ _p}$ (for the same reason as above), regardless of whether q is correct. Thus $E(c_0)$ is again nite. The argument that S_2 is satis ed is the same as before.

Now suppose $_p = 1$ or $_q = 0$. These cases are somewhat analogous to the ones above, except we need a receiver-driven protocol. Consider a protocol SR_2 in which q queries p in every round until it gets a message from p. More precisely, let req denote a request message. q sends req to p every time unit until it receives m and p sends m every time it receives req. SR_2 starts when q sends the rst req and q nishes RECEIVing m when q receives m for the rst time. By reasoning similar to the previous cases, we can show that E (# -send) and E (t-wait) are both nite (so S_3 is satis ed) and that S_2 is satis ed.

We now turn to the negative result. It turns out that the negative result is much more general than the positive result. In particular, it holds for any cost function with a certain property. In the following, we use $g \overset{1}{=}) \ f$ to denote that if $g(x) = 1$ then $f(x) = 1$.

Lemma 3.3: Let $c(r)$ be a cost function such that t-wait$(r) \overset{1}{=})\ c(r)$ and # -send$(r) \overset{1}{=})\ c(r)$. If $0 < \ _p < 1$ and $0 < \ _q < 1$, then for any send-receive protocol SR, $Pr(fr : c(r) = 1\ g) > 0$.

Proof: Suppose SR is a send-receive protocol for p and q. Let $R_1 = fr : q$ crashes at time 0 and p is correct in rg. Note that p will do the same thing in all runs in R_1: Either p stops sending after some time t or p never stops sending. If p never stops, then # -send$(r) = 1$ for all $r\ 2\ R_1$. Since, by assumption, # -send$(r) \overset{1}{=})\ c(r)$, we have that $c(r) = 1$ for each $r\ 2\ R_1$. Since $Pr(R_1) = \ _p(1 \quad _q)\ _q > 0$, we are done. Now suppose p stops sending after time t. Let $R_2 = fr : p$ crashes at time 0 and q is correct in rg. Note that q will do the same thing in all runs of R_2: Either q stops sending after some time t^0 or q never stops sending. If q never stops, then $c(r) = 1$ for all $r\ 2\ R_2$ and $Pr(R_2) = \ _q(1 \quad _p)\ _p > 0$, so again we are done. Finally, suppose that q stops sending at time t^0 in runs of R_2. Let $t^{00} = 1 + \max ft; t^0 g$. Consider $R_3 = fr :$ both processes are correct and all messages up to time t^{00} are lost in rg. Then t-wait$(r) = 1$ for all $r\ 2\ R_3$. By assumption, t-wait$(r) \overset{1}{=})\ c(r)$, so $c(r) = 1$ for all $r\ 2\ R_3$. Let n_p and n_q be the number of invocations of send by p and q, respectively, in runs of R_3 (note that p and q do the same thing in all runs of R_3). Then $Pr(R_3) = \ _p\ _q\ ^{n_p + n_q} > 0$, completing the proof. ▮ (Lemma 3.3)

Clearly # -send$(r) \overset{1}{=})\ c_0(r)$ and t-wait$(r) \overset{1}{=})\ c_0(r)$, so Lemma 3.3 applies immediately and we are done. ▮ (Theorem 3.2)

Of course, once we think in terms of utility-based speci cations like S_3, we do not want to know just whether a protocol implements S_3; we are in a position to compare the performance of di erent protocols that implement S_3 (or of variants of one protocol that all implement S_3) by considering their expected utility. Let SR_s and SR_r be generalizations (in the sense that they send messages every rounds, where need not be 1) of the sender-driven and receiver-driven protocols from Theorem 3.2, respectively. Let SR_{tr} denote the trivial (i.e., \do nothing") protocol. We use E^{SR} to denote the expectation operator determined by the probability measure on runs induced by using protocol SR. Thus, for example, E^{SR_s}(# -send) is the expected number of messages sent by SR_s. If $_p = \ _q = 0$, then SR_s, SR_r, and SR_{tr} all satisfy S_3 (although SR_{tr} does not satisfy S_2). Which is better?

In practice, process failures and link failures are very unlikely events. We assume in the rest of the paper that $_p$, $_q$, and are all very small, so that we can ignore sums of products of these terms (with coe cients like $2\ ^2$, , etc.). One way to formalize this is to say that products involving $_p$, $_q$, and are $O(")$ terms and $2\ ^2$, , etc., are $O(1)$ terms. We write $t_1 \quad t_2$ if $jt_1 \quad t_2 j$ is $O(")$. Note that we do not assume expressions like $\frac{p}{q}$ and $\frac{q}{p}$ are small.

For the following result only, we assume that not only are $_p$ and $_q$ $O(")$, they are also $(")$,[4] so that if $\frac{1}{p}$ or $\frac{1}{q}$ is multiplied by an expression that is $O("^2)$, then the result is $O(")$, which can

[4]Recall that x is $(")$ i x is $O(")$ and x^1 is $O("^{\ 1})$.

then be ignored.

Proposition 3.4: If $_p = \ _q = 0$, then

$$E^{SR_{tr}}(\text{t-wait}) = \frac{1\ (_p + \ _q \ \ _p\ _q)}{_p + \ _q \ \ _p\ _q}, \qquad E^{SR_{tr}}(\text{\#-send}) = 0,$$

$$E^{SR_s}(\text{t-wait}) \quad , \qquad E^{SR_s}(\text{\#-send}) \quad \frac{(\ +1)\ _q}{_p} + 2\frac{\frac{1}{2}\ m}{1\ m},$$

$$E^{SR_r}(\text{t-wait}) \quad 2\ , \qquad E^{SR_r}(\text{\#-send}) \quad \frac{(\ +1)\ _p}{_q} + 2\frac{}{2}\ .$$

Proof: The relatively straightforward (but tedious!) calculations are relegated to the appendix. ∎

Note that the expected cost of messages for SR_s is the same as that for SR_r, except that the roles of $_p$ and $_q$ are reversed. The expected time cost of SR_r is roughly higher than that of SR_s, because q cannot nish RECEIVing m before time 2 with a receiver-driven protocol, whereas q may nish RECEIVing m as early as with a sender-driven protocol. This says that the choice between the sender-driven and receiver-driven protocol should be based largely on the relative probability of failure of p and q. It also suggests that we should take very large to minimize costs. (Intuitively, the larger is, the lower the message costs in the case that q crashes before acknowledging p's message.) This conclusion (which may not seem so reasonable) is essentially due to the fact that we are examining a single invocation of SR in isolation. As we shall see in Section 5, this conclusion is no longer justi ed once we consider repeated invocations of SR. Finally, note that if the cost of messages is high and waiting is cheap, the processes are better o (according to this cost function) using SR_{tr}.

Thus, as far as S_3 is concerned, there are times when SR_{tr} is better than SR_s or SR_r. How much of a problem is it that SR_{tr} does not satisfy S_2? Our claim is that if this desideratum (i.e., S_2) is important, then it should be re ected in the cost function. While the cost function in our example does take into account waiting time, it does not penalize it su ciently to give us S_2. It is not too hard to nd a cost function that captures S_2. For example, suppose we take $c_1(r) = N^{\text{t-wait}(r)}$, where $N(1 \ \ _p \ \ _q + \ _p\ _q) > 1$.

Proposition 3.5: Under cost function c_1, S_3 implies S_2.

Proof: Suppose SR is a protocol that does not satisfy S_2; we show it does not satisfy S_3 (under cost function c_1). Let $C_p(t)$ and $C_q(t)$ consist of those runs of SR where p and q, respectively, are up for t time units after the start of SR (and perhaps longer). Let $R_q(t)$ consist of the runs of SR where q nishes RECEIVing m no later than time t units after the start of SR. Since SR does not satisfy S_2, there exists $" > 0$ and an increasing in nite sequence of times $t_0; t_1; :::$, such that $Pr(\overline{R_q(t_i)}\ jC_p(t_i) \setminus C_q(t_i)) > "$ for all i. We consider the case $_p = \ _q = 1$ and $_p\ _q < 1$ separately.

Suppose $_p = \ _q = 1$. Then $Pr(C_p(t) \setminus C_q(t)) = 1$ for all t. So

$$Pr(\text{t-wait} > t_i) = Pr(\overline{R_q(t_i)}) = Pr(\overline{R_q(t_i)}\ jC_p(t_i) \setminus C_q(t_i)) > "$$

for all i. Let $V_i = fr : \text{t-wait}(r) > t_i g$ and $V_1 = fr : \text{t-wait}(r) = 1\ g$. Note that $V_1 = \ ^T\ {}^1_{i=0} V_i$ and that $V_i \quad V_{i^0}$ for $i^0 > i$. Thus $Pr(V_1\ ) = Pr(\ ^T\ {}^1_{i=0} V_i) > "$. So $E(c_1) \quad Pr(V_1\ )N^1 = 1$.

Now we turn to the case that $\underline{\ _p\ _q} < 1$. Let $W(t) = fr : \text{t-wait}(r) = tg$. Note that $\text{t-wait}(r) = t_i + 1$ for all runs $r\ 2\ \overline{R_q(t_i)} \setminus \overline{C_p(t_i + 1)} \setminus C_p(t_i) \setminus C_q(t_i)$. Thus,

$$Pr(W(t_i + 1)\ jC_p(t_i) \setminus C_q(t_i)) \quad Pr(\overline{C_p(t_i + 1)} \setminus \overline{R_q(t_i)}\ jC_p(t_i) \setminus C_q(t_i)):$$

Given our independence assumptions regarding process failures,

$$\begin{aligned} Pr(\overline{C_p(t_i + 1)} \setminus \overline{R_q(t_i)}\ jC_p(t_i) \setminus C_q(t_i)) &= Pr(\overline{C_p(t_i + 1)}\ jC_p(t_i))Pr(\overline{R_q(t_i)}\ jC_p(t_i) \setminus C_q(t_i)) \\ &> (1 \quad _p)\ _p": \end{aligned}$$

A similar argument (exchanging the roles of C_p and C_q) shows that

$$\Pr(W(t_i+1) \mid C_p(t_i) \cap C_q(t_i)) \ge (1-\alpha_q)\alpha_q\epsilon.$$

So

$$\begin{aligned} E(c_1) &\ge \sum_{k=0}^{\infty} \Pr(W(k))N^{k} \\ &\ge \sum_{i=0}^{\infty} \Pr(W(t_i+1))N^{t_i+1} \\ &\ge \sum_{i=0}^{\infty} \Pr(W(t_i+1) \cap C_p(t_i) \cap C_q(t_i))N^{t_i+1} \\ &= \sum_{i=0}^{\infty} \Pr(W(t_i+1) \mid C_p(t_i) \cap C_q(t_i))\Pr(C_p(t_i) \cap C_q(t_i)))N^{t_i+1} \\ &\ge \max\{(1-\alpha_p)\alpha_p, (1-\alpha_q)\alpha_q\}\epsilon \sum_{i=0}^{\infty} (1-\alpha_p-\alpha_q+\alpha_p\alpha_q)^{t_i}N^{t_i+1}. \end{aligned}$$

Since $(1-\alpha_p-\alpha_q+\alpha_p\alpha_q)N \ge 1$ by assumption, we are done. ∎

The moral here is that S_3 gives us the flexibility to specify what really matters in a protocol, by appropriately describing the cost function. We would like to remind the reader that the cost functions are not ours to choose: They reflect the user's preferences. (Thus we are not saying that c_1 is better than c_0 or vice versa, since each user is entitled to her own preferences.) What we are really saying here is that if S_2 matters to the user, then her cost function would force S_3 to imply S_2 — in particular, her cost function could not be c_0.

4 Using Heartbeats

We saw in Section 3 that S_3 is not implementable if we are not certain about the correctness of the processes (i.e., if the probability that they are correct is strictly between 0 and 1) and the cost function $c(r)$ has the property that $\#\text{-send}(r) = \infty \Rightarrow c(r) = \infty$ and $\text{t-wait}(r) = \infty \Rightarrow c(r) = \infty$. Aguilera, Chen, and Toueg [ACT97] (ACT from now on) suggest an approach that circumvents this problem, using heartbeat messages. Informally, a heartbeat from process i is a message sent by i to all other processes to tell them that it is still alive. ACT show that there is a protocol using heartbeats that achieves quiescent reliable communication; i.e., in every run of the protocol, only finitely many messages are required to achieve reliable communication (not counting the heartbeats). Moreover, they show that, in a precise sense, quiescent reliable communication is not possible if we are not certain about the correctness of the processes and communication is unreliable, a result much in the spirit of the negative part of Theorem 3.2.[5] In this section, we show that (using the cost function c_0) we can use heartbeats to implement S_3 for all values of α_p and α_q.

For the purposes of this paper, assume that processes send a message we call hbmsg to each other every Δ time units. Protocol SR_{hb} in Figure 1 is a protocol for reliable communication based on ACT's protocol. (It is not as general as theirs, but it retains all the features relevant to us.) Briefly, what happens according to this protocol is that the failure detector layer of q sends hbmsg to the corresponding layer of p periodically. If p wants to SEND m, p checks to see if any (new)

[5] ACT actually show that their impossibility result holds even if there is only one process failure, only finitely many messages can be lost, and the processes have access to $\mathcal{S}$ (a strong failure detector), which means that eventually every faulty process is permanently suspected and at least one correct process is never suspected. The model used by ACT is somewhat different from the one we are considering, but we can easily modify their results to fit our model.

The sender's protocol (SEND):

```
1. while :receive(ack(m)) do
2.    if receive(hbmsg) then
3.       send(m)
4.
5. od
```

The receiver's protocol (RECEIVE):

```
1. while true do
2.    if receive(m) then
3.       send(ack(m))
4.
5. od
```

Figure 1: Protocol SR_{hb}

hbmsg has arrived; if so, p sends m to q, provided it has not already received ack(m) from q; q sends ack(m) every time it receives m and q nishes RECEIVing m the rst time it receives m. Note that q does not send any hbmsgs as part of SR_{hb}. That is the job of the failure-detection layer, not the job of the protocol. (We assume that the protocol is built on top of a failure-detection service.) The cost function of the previous section does ot count the costs of hbmsgs. That is, since #-send(r) is the number of messages sent by the protocol, $c_0(r)$ is not a ected by the number of hbmsgs sent in run r. It is also worth noting that this is a sender-driven protocol, quite like that given in the proof of Theorem 3.2.[6] It is straightforward to also design a receiver-driven protocol using heartbeats.

We now want to show that SR_{hb} implements S_3 and get a good estimate of the actual expected cost.

Theorem 4.1: Under cost function c_0, Protocol SR_{hb} satis es S_3. Moreover, $E(\text{t-wait}) \quad 2$ and $E(\#\text{-send}) \quad 2 \frac{2}{\quad}$, so that $E(c_0) \quad 2\ \text{c-wait} + 2 \frac{2}{\quad}$ c-send.

Proof: Using arguments similar to those of the proof of Proposition 3.4, we can show that $E(\text{t-wait}) \quad 2$ and $E(\#\text{-send}) \quad 2 \frac{2}{\quad}$. We leave details to the reader. ∎

The analysis of SR_{hb} is much like that of SR_s in Proposition 3.4. Indeed, in the case that $_p = \ _q = 0$, the two protocols are almost identical. The waiting time is roughly more for SR_{hb}, since p does not start sending until it receives the rst hbmsg from q. On the other hand, we are better o using SR_{hb} if q crashes before acknowledging p's message. In this case, with SR_s, p continues to send until it crashes, while with SR_{hb}, it stops sending (since it does not get any hbmsgs from q). This leads to an obvious question: Is it really worth sending heartbeats? Of course, if both $_p$ and $_q$ are between 0 and 1, we need heartbeats or something like them to get around the impossibility result of Theorem 3.2. But if $_p = \ _q = 0$, then we need to look carefully at the relative size of c-send and c-wait to decide which protocol has the lower expected cost.

This suggests that the decision of whether to implement a heartbeat layer must take probabilities and utilities seriously, even if we do not count either the overhead of building such a layer or the cost of heartbeats. What happens if we take the cost of heartbeats into account? This is the subject of the next section.

[6] The reader might notice that the runs induced by this protocol actually resemble those of the receiver-driven protocol in the proof of Theorem 3.2 (if we identify hbmsg with req). The di erence is that in the receiver-driven protocol in the proof of Theorem 3.2, the protocol for the receiver actually sends the reqs whereas here the hbmsgs are sent not by the protocol but by an underlying heartbeat layer, independent of the protocol.

5 The Cost of Heartbeats

In the previous section we showed that S_3 is achievable with the help of heartbeats. When we computed the expected costs, however, we did so with the cost function c_0, which does not count the cost of heartbeats. While someone who takes the heartbeat layer for granted (such as an application programmer or end-user) may have c_0 as their cost function, someone who has to decide whether to implement a heartbeat layer or how frequently heartbeats should be sent (such as a system designer) is likely to have a di erent cost function| one which takes the cost of heartbeats into account.

As evidence of this, note that it is immediate from Theorem 4.1 that under the cost function c_0, the choice of that minimizes the expected cost is clearly at most $2 + 1$. Intuitively, if we do not charge for heartbeats, there is no incentive to space them out. On the other hand, if we do charge for heartbeats, then typically we will be charging for heartbeats that are sent long after a given invocation of SR_{hb} has completed.

The whole point of having a heartbeat layer is that heartbeats are meant to be used, not just by one invocation of a single protocol, but by multiple invocations of (possibly) many protocols. We would expect that the optimal frequency of heartbeats should depend in part on how often the protocols that use them are invoked. The picture we have is that the SR_{hb} protocol is invoked from time to time, by di erent processes in the system. It may well be that various invocations of it are running simultaneously. All these invocations share the heartbeat messages, so their cost can be spread over all of them. If invocations occur often, then there will be few \wasted" heartbeats between invocations, and the analysis of the previous subsection gives a reasonably accurate reading of the costs involved. On the other hand, if is small and invocations are infrequent, then there will be many \wasted" heartbeats. We would expect that if there are infrequent invocations, then heartbeats should be spaced further apart.

We now consider a setting that takes this into account. For simplicity, we continue to assume that there are only two processes, p and q, but we now allow both p and q to invoke SR_{hb}. (It is possible to do this with n processes and more than one protocol, but the two-process and single protocol case su ces to illustrate the main point, which is that the optimal should depend on how often the protocol is invoked.) We assume that each process, while it is running, invokes SR_{hb} with probability at each time unit. Thus, informally, at every round, each running process tosses a coin with probability of of landing heads. If it lands heads, the process then invokes SR_{hb} with the other as the recipient. (Note that we no longer assume that the protocol is invoked at time 0 in this section.)

Roughly speaking, in computing the cost of a run, we consider the cost of each invocation of SR_{hb} together with the cost of all the heartbeat messages sent in the run. Our interest will then be in the cost per invocation of SR_{hb}. Thus, we apportion the cost of the heartbeat messages among the invocations of SR_{hb}. If there are relatively few invocations of SR_{hb}, then there will be many \wasted" heartbeat messages, whose cost will need to be shared among them.

For simplicity, let us assume that each time SR_{hb} is invoked, a di erent message is sent. (For example, messages could be numbered and include the name of the sender and recipient.) We say $SR_{hb}(m)$ is invoked at time t_1 in r if at time t_1 some process x rst executes line 1 of the code of the sender with message m. This invocation of SR_{hb} completes at time t_2 if the last message associated with the invocation (either a copy of m or a copy of $ack(m)$) is sent at time t_2. If x received the last heartbeat message from the receiver before invoking $SR_{hb}(m)$, we take $t_2 = t_1$ (that is, the invocation completes as soon as it starts in this case).

The processes will (eventually) stop sending m or $ack(m)$ if either process crashes or if the sender receives $ack(m)$. Thus, with probability 1, all invocations of SR_{hb} will eventually complete.

Let #-SR(r,t) be the number of invocations of SR$_{hb}$ that have completed by time t in r; let c-SR(r,t) be the cost of these invocations. Let c-hbmsg(r,t) be the cost of sending hbmsg up to time t in r. This is simply the number of hbmsgs sent up to time t (which we denote by #-hbmsg(r,t)) multiplied by c-send. Let $c^{total}(r,t) =$ c-SR(r,t) + c-hbmsg(r,t). Finally, let

$$c^{avg}(r) = \lim\sup_{t\to\infty} \frac{c^{total}(r,t)}{\#\text{-SR}(r,t)+1},$$

where "lim sup" denotes the limit of the supremum, that is,

$$c^{avg}(r) = \lim_{t'\to\infty} \sup_{0\ \ t\ \ t'} \frac{c^{total}(r,t)}{\#\text{-SR}(r,t)+1}.^{7}$$

Thus $c^{avg}(r)$ is essentially the average cost per invocation of SR$_{hb}$, taking heartbeats into account. We write "lim sup" instead of "lim" since the limit may not exist in general. (However, the proof of the next theorem shows that in fact, with probability 1, the limit does exist.) For the following result only, we assume that $\sqrt{\ _p}$ and $\sqrt{\ _q}$ are also $O(")$.

Theorem 5.1: Under the cost function c^{avg}, Protocol SR$_{hb}$ satisfies S_3. Furthermore, $E(c^{avg})$ $((1\ \ _p)(1\ \ _q) + \ _p\ _q)\ 2\ \frac{2}{\ }$ c-send + $\ \ + \frac{1}{2}$ c-wait + $\frac{1}{\ }$c-send, where $0 < \ \ < 1$.

Proof: See the appendix.∎

Note that with this cost function, we have a real decision to make in terms of how frequently to send heartbeats. As before, there is some benefit to making $\ > 2\ $: it minimizes the number of redundant messages sent when SR$_{hb}$ is invoked (that is, messages sent by the sender before receiving the receiver's acknowledgment). Also, by making larger we will send fewer heartbeat messages between invocations of SR$_{hb}$. On the other hand, if we make too large, then the sender may have to wait a long time after invoking SR$_{hb}$ before it can send a message to the receiver (since messages are only sent upon receipt of a heartbeat). Intuitively, the greater c-wait is relative to c-send, the smaller we should make . Clearly we can find an optimal choice for by standard calculus.

In the model just presented, if c-wait is large enough relative to c-send, we will take to be 1. Taking this small is clearly inappropriate once we consider a more refined model, where there are buffers that may overflow. In this case, both the probability of message loss and the time for message delivery will depend on the number of messages in transit. The basic notions of utility still apply, of course, although the calculations become more complicated. This just emphasizes the obvious point is that in deciding what value (or values) should have, we need to carefully look at the actual system and the cost function.

6 Discussion

We have tried to argue here for the use of decision theory both in the specification and the design of systems. Our (admittedly rather simple) analysis already shows both how decision theory can help guide the decision made and how much the decision depends on the cost function. None of our results are deep; the cost function just makes precise what could already have been seen from an intuitive calculation. But this is precisely the point: By writing our specification in terms of costs, we can make the intuitive calculations precise. Moreover, the specification forces us to make clear

[7] By adding 1 to the denominator, we guarantee it is never 0; adding 1 also simplifies one of the technical calculations needed in the proof of Theorem 5.1.

exactly what the cost function is and encourages the elicitation of utilities from users. We believe that these are both important features. It is important for the user (and system designer) to spend time thinking about what the important attributes of the system are and to decide on preferences between various tradeoffs.

A possible future direction is to study standard problems in the literature (e.g., Consensus, Byzantine Agreement, Atomic Broadcast, etc.) and recast the specifications in utility-theoretic terms. One way to do this is to replace a liveness requirement by an unbounded increasing cost function (which is essentially the “cost of waiting”) and replace a safety requirement by a large penalty. Once we do this, we can analyze the algorithms that have been used to solve these problems, and see to what extent they are optimal given reasonable assumptions about probabilities and utilities.

While we believe that there is a great deal of benefit to be gained from analyzing systems in terms of utility, it is quite often a nontrivial matter. Among the most significant difficulties are the following:

1. Where are the utilities coming from? It is far from clear that a user can or is willing to assign a real-valued utility to all possible outcomes in practice. There may be computational issues (for example, the set of outcomes can be enormous) as well as psychological issues. While the agent may be prepared to assign qualitative utilities like “good”, “fair”, or “bad”, he may not be prepared to assign 20.7. While to some extent the system can convert qualitative utilities to a numerical representation, this conversion may not precisely captures the user's intent. There are also nontrivial user-interface issues involved in eliciting utilities from users. In light of this, we need to be very careful if results depend in sensitive ways on the details of the utilities.

2. Where are the probabilities coming from? We do not expect users to be experts at probability. Rather, we expect the system to be gathering statistics and using them to estimate the probabilities. Of course, someone still has to tell the system what statistics to gather. Moreover, our statistics may be so sparse that we cannot easily obtain a reliable estimate of the probability.

3. Why is it even appropriate to maximize expected utility? There are times when it is far from clear that this is the best thing to do, especially if our estimates of the probability and utility are suspect. For example, suppose one action has a guaranteed utility of 100 (on some appropriate scale), while another has an expected utility of 101, but has a nontrivial probability of having utility 0. If the probabilities and utilities that were used to calculate the expectation are reliable, and we anticipate performing these actions frequently, then there is a good case to be made for taking the action with the higher expected utility. On the other hand, if the underlying numbers are suspect, then the action with the guaranteed utility might well be preferable.

We see these difficulties not as ones that should prevent us from using decision theory, but rather as directions for further research. It may be possible in many cases to learn a user's utility. Moreover, we expect that in many applications, except for a small region of doubt, the choice of which decision to make will be quite robust, in that perturbations to the probability and utility will not change the decision. Even in cases where perturbations do change the decision, both decisions will have roughly equal expected utility. Thus, as long as we can get somewhat reasonable estimates of the probability and utility, decision theory may have something to offer.

Another important direction for research is to consider qualitative decision theory, where both utility and likelihood are more qualitative, and not necessarily real numbers. This is, in fact, an active area of current research, as `http://www.medg.lcs.mit.edu/qdt/bib/unsorted.bib` (a bibliography of over 290 papers) attests. Note that once we use more qualitative notions, then we may not be able to compute expected utilities at all (since utilities may not be numeric) let alone take the action with maximum expected utility, so we will have to consider other decision rules.

Finally, we might consider what would be an appropriate language to specify and reason about utilities, both for the user and the system designer.

While it is clear that there is still a great deal of work to be done in order to use decision-theoretic techniques in systems design and speci cation, we hope that this discussion has convinced the reader of the utility of the approach.

Acknowledgments

We thank Sam Toueg for numerous discussions regarding heartbeats and Jim Gray for giving us some insight on costs in database computations and for pointing out the use of costs in deadlock detection. We also thank the anonymous referees for their helpful comments.

Appendix: Proofs

We present the proofs of Proposition 3.4 and Theorem 5.1. We repeat the statements of the results for the convenience of the reader. Recall that for Proposition 3.4, we are assuming that $_p$ and $_q$ are both ("), and that for Theorem 5.1, we are assuming that $\frac{p}{\ \ }_p$ and $\frac{p}{\ \ }_q$ are both O (").

Proposition 3.4: If $_p = \ _q = 0$, then

$$\begin{aligned} E^{SR_{tr}}(\text{t-wait}) &= \frac{1 \ (_p + \ _q \ \ _p \ _q)}{_p + \ _q \ \ _p \ _q}, & E^{SR_{tr}}(\#\text{-send}) &= 0,\\ E^{SR_s}(\text{t-wait}) & \ \ , & E^{SR_s}(\#\text{-send}) & \ \frac{(\ +1)\ _q}{_p} + 2 \left\lceil\frac{1}{2}\right\rceil^{m},\\ E^{SR_r}(\text{t-wait}) & \ \ 2 \ , & E^{SR_r}(\#\text{-send}) & \ \frac{(\ +1)\ _p}{_q} + 2 \left\lceil\frac{1}{2}\right\rceil^{m}. \end{aligned}$$

Proof: For SR_{tr}, note that #-send(r) = 0 for all r, so $E^{SR_{tr}}(\#\text{-send}) = 0$. We also have that t-wait(r) is the time of the rst crash in r. Since the probability of a crash during a time unit is $= \ _p + \ _q \ \ _p \ _q$, we have that the expected time of the rst crash, and hence $E^{SR_{tr}}(\text{t-wait})$, is

$$\sum_{k=0}^{1} k(1 \ \ )^k = \frac{(1 \ \ )}{(1 \ \ (1 \ \ ))^2} = \frac{1}{\ \ } = \frac{1 \ (_p + \ _q \ \ _p \ _q)}{_p + \ _q \ \ _p \ _q}:$$

For SR_s, we rst show that $E^{SR_s}(\text{t-wait})$. Since $_p = \ _q = 0$, $Pr(\text{t-wait}(r) = 1\ ) = 0$, thus $E^{SR_s}(\text{t-wait}) = \sum_{k=1}^{1} k\,Pr(\text{t-wait} = k)$. We break the sum into three pieces,

$$\sum_{k=1}^{1} k\,Pr(\text{t-wait} = k),$$

$$Pr(\text{t-wait} = \ \ ), \text{and}$$

$$\sum_{k=\ +1}^{1} k\,Pr(\text{t-wait} = k),$$

and analyze each one separately.

For the rst part, note that the only way that t-wait = k for 1 k < is for there to be a crash before . Thus

$$\Pr(\text{t-wait} = k) = ((1 \quad _p)(1 \quad _q))^k (_p + \quad _q \quad _p \, _q) < \quad _p + \quad _q:$$

It follows that

$$\sum_{k=1}^{1} k \Pr(\text{t-wait} = k) < (_p + \quad _q) \sum_{k=1}^{1} k = (_p + \quad _q)\frac{(\quad 1)}{2} \qquad 0:$$

Thus we may drop the rst part.

For the second part, note that t-wait = if p and q are up until and q received the rst copy of m p sent. (We may also have t-wait = if one of p or q crashes at time .) Thus,

$$\Pr(\text{t-wait} = \quad) \quad ((1 \quad _p)(1 \quad _q)) \; (1 \quad) \quad 1;$$

so the second part is .

Finally, for the third part, if k > , then k has the form + a + b, where a 0 and 0 b < (and a + b > 0). If t-wait = k = + a + b, then a + 1 messages are lost by the link, so $\Pr(\text{t-wait} = k) \quad ^{a+1}$. A straightforward calculation shows that

$$\begin{aligned}
\sum_{k= \; +1}^{1} k \Pr(\text{t-wait} = k) &= \sum_{b=1}^{1} (\; + b) \Pr(\text{t-wait} = \; + b) \\
&\quad + \sum_{a=1}^{1} \sum_{b=0}^{1} (\; + a \; + b) \Pr(\text{t-wait} = \; + a \; + b) \\
&\quad \sum_{a=0}^{1} (\; + (a+1) \;) \quad ^{a+1} \\
&\quad \sum_{a=0}^{1} ((a+1) \; ^2 + \quad) \quad ^{a+1} \\
&= \quad ^2 \sum_{a=1}^{1} a \quad ^a + \sum_{a=1}^{1} \quad ^a \\
&\quad 0:
\end{aligned}$$

Thus, we can also ignore the third part. This gives us E^{SR_s}(t-wait)) , as desired.

Now let us turn to E^{SR_s}(# -send). Let us say that a send is successful i the link does not drop the message (which could be an ack). Consider the set of runs A = fr : q successfully sends ack(m) before crashing in rg. Roughly speaking, what happens is that in runs of A, p is receives ack(m) at time 2 with probability $\frac{1}{m}$ 1. In the meantime, p has sent m exactly $\frac{2}{}$ times with probability 1. With probability $\frac{1}{m}$ 1, all of these are received by q; q in turn acknowledges all copies and thus E^{SR_s}(# -send jA) 2 $\frac{2}{}$; that is why this term appears in E^{SR_s}(# -send). In $\overline{A}$, the expected value of # -send is very large, since p will send m until it crashes, so despite the low probability of $\overline{A}$, it contributes the term $\frac{(\; +1) \; _q}{_p}$. We now turn to the details.

We rst compute Pr(A). Note that q can send ack(m) only at times of the form + k . Let B_k = fr : q sends the rst successful ack(m) at time + k g. Note that $A = \bigcup_{k=0}^{1} B_k$ and that $B_i \backslash B_j$ = ; if i 6= j. Thus $\Pr(A) = \sum_{k=0}^{1} \Pr(B_k)$. Since q sends the rst successful ack(m) at time + k in runs of B_k, p must (successfully) send m at time k in runs of B_k. Thus

$$\Pr(B_k) = (1 \quad _p)^{k \; +1} (1 \quad _q)^{\; + k \; +1} (2 \quad \; ^2)^k (1 \quad)^2:$$

The rst factor re ects the fact that p must have been up at time k (to send m) while the second factor re ects the fact that q must have been up at time + k (to receive m and send ack(m)). The third factor re ects the fact that the previous k attempts have failed: either m was lost or the corresponding ack(m) was lost, which occurs with probability (+ (1)) = 2 ^{2}. The nal factor re ects the fact that the (k + 1)st attempt succeeded: both messages got through. So

$$\begin{aligned}
\Pr(A) &= \sum_{k=0}^{\infty} \Pr(B_k) \\
&= \sum_{k=0}^{\infty} (1 \quad {}_p)^{k+1}(1 \quad {}_q)^{\ +k+1}(2 \quad {}^2)^k(1 \quad)^2 \\
&= (1 \quad {}_p)(1 \quad {}_q)^{\ +1}(1 \quad)^2 \sum_{k=0}^{\infty} (1 \quad {}_p)^k (1 \quad {}_q)^k (2 \quad {}^2)^k \\
&= (1 \quad {}_p)(1 \quad {}_q)^{\ +1}(1 \quad)^2 \frac{1}{1 \quad (1 \quad {}_p)\ (1 \quad {}_q)\ (2 \quad {}^2)} \\
&= (1 \quad {}_p)(1 \quad {}_q)^{\ +1}(1 \quad)^2((1 + 2\ ) + O(\varepsilon^2)) \\
&= 1 \quad {}_p \quad (\ +1)\ {}_q + O(\varepsilon^2) \\
&\quad 1:
\end{aligned}$$

We now want to compute E^{SR_s}(#-send$|A$). Again, we break E^{SR_s}(#-send$|A$) into three pieces,

$$\sum_{k=0}^{2\lceil\frac{2}{}\rceil \ 1} k \Pr(\text{\#-send} = k|A),$$

$$2\left\lceil\frac{2}{}\right\rceil \Pr\left(\text{\#-send} = 2\left\lceil\frac{2}{}\right\rceil |A\right), \text{and}$$

$$\sum_{k=2\lceil\frac{2}{}\rceil+1}^{\infty} k \Pr(\text{\#-send} = k|A),$$

and compute each part separately.

Note that Pr(#-send = k$|$A) ${}_p$ + ${}_q$ + for $k < 2\left\lceil\frac{2}{}\right\rceil$, since either a process crashed or a message is lost. Thus the rst part is no more than $2\left\lceil\frac{2}{}\right\rceil$ (${}_p$ + ${}_q$ +) 0, so we may ignore it. For the second part, we have

$$\Pr\left(\text{\#-send} = 2\left\lceil\frac{2}{}\right\rceil |A\right) \quad (1 \quad {}_p)^{2\ +1}(1 \quad {}_q)^{\lceil\frac{2}{}\rceil+\ +1}(1 \quad)^{\lceil\frac{2}{}\rceil+1} \quad 1;$$

since if p is up at time 2 , q is up at time $\left\lceil\frac{2}{}\right\rceil$ + , all of p's sends got through, and q's rst ack(m) got through, then #-send = $2\left\lceil\frac{2}{}\right\rceil$; thus the second part is $2\left\lceil\frac{2}{}\right\rceil$. We now turn our attention to the last part.

Note that p sends at least half the messages in every run r (whether $r \in A$ or $r \in \overline{A}$). Note also that, after the rst successful attempt (that is, after the rst message sent by p which is received by q whose corresponding acknowledgment is not lost by the link), p will send at most $\left\lceil\frac{2}{}\right\rceil$ messages, since p would stop sending 2 time units after the rst successful attempt (either because p received ack(m) or p crashed). Combining the above two observations, we see that if #-send(r) = $2\left\lceil\frac{2}{}\right\rceil + k$ for $k > 0$, then p must have sent at least $\left\lceil\frac{2}{}\right\rceil + \left\lceil\frac{k}{2}\right\rceil$ messages and there are at least $\left\lceil\frac{k}{2}\right\rceil$ unsuccessful attempts in r. Thus, $\Pr(\#\text{-send} = 2\frac{1\quad m}{2} + k \mid A) \quad (2 \qquad {}^2)^{\lceil\frac{k}{2}\rceil}$. So we have

$$\begin{aligned}\sum^{1}_{k=2\lceil\frac{2}{}\rceil+1} k\Pr(\#\text{-send} = k \mid A) &\quad \sum^{1}_{k=2\lceil\frac{2}{}\rceil+1} k(2 \qquad {}^2)^{\lceil\frac{k}{2}\rceil}\\ &= \sum^{1}_{k=\lceil\frac{2}{}\rceil} ((2k+1) + (2k+2))(2 \qquad {}^2)^{k+1}\\ &= \sum^{1}_{k=\lceil\frac{2}{}\rceil} (4k+3)(2 \qquad {}^2)^{k+1}\\ &\quad 0:\end{aligned}$$

So we may ignore the last part as well. Thus $E^{SR_s}(\#\text{-send} \mid A) \quad 2\frac{1\quad m}{2}$. Since $\Pr(A) \quad 1$, we have $E^{SR_s}(\#\text{-send} \mid A)\Pr(A) \quad 2\frac{1\quad m}{2}$.

We now focus on $E^{SR_s}(\#\text{-send} \mid \overline{A})\Pr(\overline{A})$. Recall that for $r \in \overline{A}$, q fails to successfully send ack(m) in r. Consider the following three sets (which is a partition of the set of all runs):

$C_1 = \{r : \text{p crashes at time 0 in } r\}$,

$C_2 = \{r : \text{p does not crash at time 0 and q crashes at or before time} \quad \text{in } r\}$, and

$C_3 = \{r : \text{p does not crash at time 0 and q does not crash at or before time} \quad \text{in } r\}$.

We now show that these are their probabilities:

$\Pr(C_1 \cap \overline{A}) = \quad {}_p$,

$\Pr(C_2 \cap \overline{A}) = (1 \quad {}_p)(1 \quad (1 \quad {}_q)^{\ +1}) = (\ +1)\ {}_q + O(\varepsilon^2)$, and

$\Pr(C_3 \cap \overline{A}) = O(\varepsilon^2)$.

First note that $\Pr(C_1) = \quad {}_p$ and $\Pr(C_2) = (1 \quad {}_p)(1 \quad (1 \quad {}_q)^{\ +1}) = (\ +1)\ {}_q + O(\varepsilon^2)$. Furthermore, $C_1 \cup C_2 \quad \overline{A}$, since if $r \in C_1 \cup C_2$, q does not send ack(m) successfully before crashing. Thus $\Pr(C_1 \cap \overline{A}) = \quad {}_p$ and $\Pr(C_2 \cap \overline{A}) = (\ +1)\ {}_q + O(\varepsilon^2)$. Since, as we showed earlier, $\Pr(A) = 1 \quad {}_p \quad (\ +1)\ {}_q + O(\varepsilon^2)$, it also follows that $\Pr(C_3 \cap \overline{A}) = O(\varepsilon^2)$.

Now that we have $\Pr(C_i \cap \overline{A})$, let us turn to $E^{SR_s}(\#\text{-send} \mid C_i \cap \overline{A})$. Note that for $r \in \overline{A}$, p will send messages until it crashes. For $r \in C_1$, p crashes immediately, so $\#\text{-send}(r) = 0$ for $r \in C_1$. For $r \in C_2$, q crashes before it can possibly send any messages, so all the messages are sent by p. Thus

$$\Pr(\#\text{-send} = k \mid C_2) = (1 \quad {}_p)^{(k\quad 1)\ +1}(1 \quad (1 \quad {}_p)\ );$$

since p must be up at time $(k \quad 1)$ and crash before time k to send m exactly k times. So

$$\begin{aligned}E^{SR_s}(\#\text{-send} \mid C_2 \cap \overline{A}) &= \sum^{1}_{k=1} k(1 \quad {}_p)^{(k\quad 1)\ +1}(1 \quad (1 \quad {}_p)\ )\\ &= \frac{(1 \quad {}_p)(1 \quad (1 \quad {}_p)\ )}{(1 \quad {}_p)} \sum^{1}_{k=1} k((1 \quad {}_p)\ )^k\\ &= \frac{(1 \quad {}_p)(1 \quad (1 \quad {}_p)\ )}{(1 \quad {}_p)} \frac{(1 \quad {}_p)}{(1 \quad (1 \quad {}_p)\ )^2}\\ &= \frac{1 \quad {}_p}{1 \quad (1 \quad {}_p)}\\ &= \frac{1}{\ {}_p} + O(1):\end{aligned}$$

The O(1) term is there because $\frac{1}{\ _p} \quad \frac{1\ \ _p}{1\ (1\ \ _p)} = \frac{1}{\ _p} \quad \frac{1\ \ _p}{\ _p + O(''^2)} = \frac{O(''^2)}{(\ _p)^2 + O(''^3)}$, which is O(1), since we assumed that $_p$ is ('') for this proposition.

For r 2 $C_3 \setminus \overline{A}$, q might send messages (none of which, however, will get through). Let E_k = fr 2 $C_3 \setminus \overline{A}$: p crashes at time kg. We have Pr(E_k) (1 $_p)^k$ $_p$. Furthermore, E^{SR_s}(# -send jE_k) 2 $\frac{k}{}$ $\frac{1}{}$ $\frac{}{m}$, since p sends $\frac{k}{}$ $\frac{1}{}$ $\frac{}{m}$ messages in E_k and q sends at most that many messages. So we have

$$\begin{aligned}
E^{SR_s}(\#\text{-send } j C_3 \setminus \overline{A}) &= \sum_{k=1}^{\infty} E^{SR_s}(\#\text{-send } j E_k) Pr(E_k) \\
& \sum_{k=1}^{\infty} 2 \frac{k}{} \frac{1}{} \frac{}{m} (1 \ \ _p)^k \ _p \\
& \sum_{k=1}^{\infty} 2 \frac{k}{} + 1 \ (1 \ \ _p)^k \ _p \\
&= \frac{2 \ _p}{} \sum_{k=1}^{\infty} k(1 \ \ _p)^k + 2 \ _p \sum_{k=1}^{\infty} (1 \ \ _p)^k \\
&= \frac{2 \ _p}{} \frac{1 \ \ _p}{\ _p^2} + 2(1 \ \ _p) \\
&= \frac{2}{\ _p} + O(1):
\end{aligned}$$

Since we assumed that $_p$ is (''), E^{SR_s}(# -send j $C_3 \setminus \overline{A}$)Pr($C_3 \setminus \overline{A}$) = O(''). Recall that E^{SR_s}(# -send j$C_1 \setminus \overline{A}$) = 0, so

$$E^{SR_s}(\#\text{-send } j\overline{A}) Pr(\overline{A}) \quad E^{SR_s}(\#\text{-send } j C_2 \setminus \overline{A}) Pr(C_2 \setminus \overline{A}) \quad \frac{(\ +1)\ _q}{\ _p}:$$

This gives us E^{SR_s}(# -send) $\frac{(\ +1)\ _q}{\ _p}$ + 2 $\frac{}{2}$ $\frac{1}{}$ $\frac{}{m}$ as desired.

The reasoning for the SR_r case is similar to the SR_s case. The only major di erence is that q cannot possibly nish RECEIVing m before time 2 . We leave details to the reader.∎

Theorem 5.1: Under the cost function c^{avg}, Protocol SR_{hb} satis es S_3. Furthermore, E(c^{avg}) ((1 $_p$)(1 $_q$) + $_p$ $_q$) 2 $\frac{}{2}$ $\frac{1}{}$ $\frac{}{m}$ c-send + + $\frac{1}{\ 2}$ c-wait + $\frac{1}{}$c-send, where 0 < < 1.

Proof: Roughly speaking, the rst summand corresponds to the expected per-invocation cost of the protocol and the second corresponds to the expected per-invocation cost of the heartbeats. To do the analysis carefully, we divide the set of runs into three subsets:

F_1 = fr : one process is correct and the other eventually crashes in rg,

F_2 = fr : both processes are correct in rg, and

F_3 = fr : both processes eventually crash in rg.

These are their probabilities:

Pr(F_1) = $_p$(1 $_q$) + $_q$(1 $_p$),

Pr(F_2) = $_p$ $_q$, and

$\Pr(F_3) = (1 \quad _p)(1 \quad _q)$.

For $r \in F_1$, we expect the lone correct process to invoke SR_{hb} infinitely often. All but finitely many of these invocations will take place after the other process crashed. Thus the average cost of an invocation in r will be 0. For $r \in F_2$, on the other hand, both processes are expected to invoke SR_{hb} infinitely often and the average cost of the invocation in r is expected to be close to the expected cost of a single invocation of SR_{hb}. The computation of the expected cost of an invocation in a run in F_3 is more delicate. We now examine the details.

Let G_1 be the subset of F_1 consisting of runs r in which the correct process tries to invoke the protocol infinitely often. Clearly $\Pr(G_1 | F_1) = 1$, since the protocol is invoked with probability at each time unit. Moreover, for each run $r \in G_1$, we have

$$\lim_{t \to \infty} \frac{c\text{-}SR(r;t)}{\#\text{-}SR(r;t) + 1} = 0;$$

since there are only finitely many complete invocations with non-zero cost and there are infinitely many complete invocations. Thus, $E(c^{avg} | F_1) = 0$.

Let G_2 be the subset of F_2 where there are infinitely many invocations of SR_{hb}. Clearly $\Pr(G_2 | F_2) = 1$. Let $Z = 2 \frac{2}{\ } c\text{-send} + \quad + \frac{1}{2} c\text{-wait}$. By the Law of Large Numbers, for almost all runs r of G_2, the analysis of Proposition 3.4 shows that

$$\lim_{t \to \infty} \frac{c\text{-}SR(r;t)}{\#\text{-}SR(r;t) + 1} \quad Z:$$

(Note that we have $\ + \frac{1}{2}$ instead of 2 as in Theorem 4.1. This is because in the current setting, the expected amount of time elapsed between the start of an invocation and the arrival of the first hbmsg is $\frac{1}{2}$. In the setting of Theorem 4.1, however, the first hbmsg cannot arrive until time , since the invocation starts at time 0 and the first hbmsg is sent at time 0. Note that in both cases, the expected time of waiting is plus the expected time elapsed between the start of the invocation and the arrival of the next hbmsg.) Thus $\Pr(c^{avg}(r) \quad Z | F_2) = 1$.

We now turn our attention to F_3. Let $F_3(t_1;t_2;i_1;i_2;i_3)$ be a subset of F_3 with the following properties:

- the first crash in r happens at time t_1,
- the second crash in r happens at time t_2,
- the number of invocations starting before time $t_1 \quad 3$ is i_1,
- the number of invocations starting between times $t_1 \quad 3$ and $t_1 +$ is i_2, and
- the number of invocations starting after time $t_1 +$ is i_3.

It is clear that each of these sets are measurable. (Some of them are empty, so they will have probability 0; we could introduce restrictions to rule out the empty ones, but leaving them in is not a problem.)

Suppose $F_3(t_1;t_2;i_1;i_2;i_3)$ is not empty. Then

$$E(c^{avg} | F_3(t_1;t_2;i_1;i_2;i_3)) \quad \frac{i_1 + \ (t_1;t_2;i_1;i_2;i_3) i_2}{i_1 + i_2 + i_3 + 1} Z;$$

where $0 < \ (t_1;t_2;i_1;i_2;i_3) < 1$. Roughly speaking, the expected cost of an invocation in the first group is Z, since if no messages are lost (which happens with probability 1), the number of messages sent is exactly $2\left\lceil\frac{}{2}\right\rceil$ and the time of waiting is between and + 1, depending on when the rst hbmsg arrives after the invocation starts. If no messages are lost, a hbmsg is received every time units, so the wait for a hbmsg is $\frac{1}{2}$ on average. Thus the rst group of invocations contribute $i_1 Z$ to c-SR(r), on average. As for the second group, they contribute something less than $i_2 Z$ to c-SR(r) on average; in many of these invocation, the rst process crash (which happens at most 3 + after the beginning of an invocation in the second group) may reduce the time of waiting or the number of messages sent. That is why we have a multiplicative constant $(t_1;t_2;i_1;i_2;i_3)$ in front of i_2. The last group of invocations all have zero cost, since by the time they started, the surviving process (which must be the invoker) will never receive any new hbmsgs from the crashed process; so the time of waiting and the number of messages sent are both zero.

Thus we have

$$E(c^{avg} \mid F_3) = \sum_{t_1;t_2;i_1;i_2;i_3} E(c^{avg} \mid F_3(t_1;t_2;i_1;i_2;i_3))\Pr(F_3(t_1;t_2;i_1;i_2;i_3))$$
$$Z \sum_{t_1;t_2;i_1;i_2;i_3} \frac{i_1 + (t_1;t_2;i_1;i_2;i_3) i_2}{i_1+i_2+i_3+1} \Pr(F_3(t_1;t_2;i_1;i_2;i_3)):$$

Let

$$= \sum_{t_1;t_2;i_1;i_2;i_3} \frac{i_1 + (t_1;t_2;i_1;i_2;i_3) i_2}{i_1+i_2+i_3+1} \Pr(F_3(t_1;t_2;i_1;i_2;i_3)):$$

Clearly < 1 and $E(c^{avg} \mid F_3)$ Z , as desired.

Now we turn to the expected heartbeat costs per invocation. Each process will send a hbmsg every time units for as long as it is up. So if in r a process is up at time t, then it sent $\left\lceil\frac{t}{}\right\rceil$ hbmsgs in r up to time t. Suppose r 2 F_2. Then, # -hbmsg(r;t) = $2\left\lceil\frac{t}{}\right\rceil$, and by the Law of Large Numbers, for all > 0,

$$\Pr\left(\lim_{t! 1} |\#\text{-SR}(r;t) \quad 2t| \quad t \,\middle|\, F_2\right) = 1:$$

Thus,

$$\Pr\left(\lim_{t! 1} \frac{\#\text{-hbmsg}(r;t)}{\#\text{-SR}(r;t)+1} = \frac{1}{} \,\middle|\, F_2\right) = 1:$$

Next, suppose r 2 F_1. Then one of the processes will send only nitely many hbmsgs and invoke SR_{hb} nitely often. Thus after the crash, we have

$$\frac{\#\text{-hbmsg}(r;t)}{\#\text{-SR}(r;t)+1} = \frac{\left\lceil\frac{t}{}\right\rceil + H}{I_2+I_1+1};$$

where H is the number of times the crashed process sends hbmsg in r, I_1 is the number of times the crashed process invoked SR_{hb} in r, and I_2 is the number of times the live process invoked SR_{hb} in r. For all > 0, we have that

$$\Pr\left(\lim_{t! 1} |I_2 \quad t| \quad t \,\middle|\, F_1\right) = 1:$$

Thus,

$$\Pr\left(\lim_{t! 1} \frac{\#\text{-hbmsg}(r;t)}{\#\text{-SR}(r;t)+1} = \frac{1}{} \,\middle|\, F_1\right) = 1:$$

Finally, consider the set F_3, where both processes crash. Again, the situation here is more complicated, since there are only finitely many complete invocations and hbmsgs in each run, so we cannot resort to the Law of Large Numbers. Let $F_3(j,k)$ be the set of runs where p crashes at time j and q crashes at time k. Clearly $\Pr(F_3(j,k)\mid F_3) = (1-\alpha_p)^j(1-\alpha_q)^k\alpha_p\alpha_q$ and the number of heartbeats sent in runs of $F_3(j,k)$ is $\lceil \frac{j}{\eta}\rceil + \lceil \frac{k}{\eta}\rceil$. Let $\#\text{-hbmsg}^{avg}(r) = \lim_{t\to\infty}\frac{\#\text{-hbmsg}(r,t)}{\#\text{-SR}(r,t)+1}$. Observe that

$$
\begin{aligned}
E(\#\text{-hbmsg}^{avg}\mid F_3(j,k)) &= \sum_{i=0}^{j+k}\frac{\lceil\frac{j}{\eta}\rceil+\lceil\frac{k}{\eta}\rceil}{i+1}\gamma^i(1-\gamma)^{j+k-i}\binom{j+k}{i}\\
&= \frac{\lceil\frac{j}{\eta}\rceil+\lceil\frac{k}{\eta}\rceil}{\gamma(j+k+1)}\sum_{i=0}^{j+k}\gamma^{i+1}(1-\gamma)^{j+k-i}\binom{j+k+1}{i+1}\\
&= \frac{\lceil\frac{j}{\eta}\rceil+\lceil\frac{k}{\eta}\rceil}{\gamma(j+k+1)}\sum_{i=1}^{j+k+1}\gamma^{i}(1-\gamma)^{j+k+1-i}\binom{j+k+1}{i}\\
&= \frac{\lceil\frac{j}{\eta}\rceil+\lceil\frac{k}{\eta}\rceil}{\gamma(j+k+1)}(1-(1-\gamma)^{j+k+1}).
\end{aligned}
$$

Thus,

$$
\begin{aligned}
E(\#\text{-hbmsg}^{avg}\mid F_3) &= \sum_{j,k}E(\#\text{-hbmsg}^{avg}\mid F_3(j,k))\Pr(F_3(j,k))\\
&= \sum_{j,k}\frac{\lceil\frac{j}{\eta}\rceil+\lceil\frac{k}{\eta}\rceil}{\gamma(j+k+1)}(1-(1-\gamma)^{j+k+1})(1-\alpha_p)^j(1-\alpha_q)^k\alpha_p\alpha_q\\
&= \sum_{j,k}\frac{\lceil\frac{j}{\eta}\rceil+\lceil\frac{k}{\eta}\rceil}{\gamma(j+k+1)}(1-\alpha_p)^j(1-\alpha_q)^k\alpha_p\alpha_q -\\
&\quad \sum_{j,k}\frac{\lceil\frac{j}{\eta}\rceil+\lceil\frac{k}{\eta}\rceil}{\gamma(j+k+1)}(1-\gamma)^{j+k+1}(1-\alpha_p)^j(1-\alpha_q)^k\alpha_p\alpha_q.
\end{aligned}
$$

Note that

$$\frac{\lceil\frac{j}{\eta}\rceil+\lceil\frac{k}{\eta}\rceil}{\gamma(j+k+1)} < L_1$$

for some constant L_1 (roughly $\frac{1}{\gamma\eta}$). Thus the second summand above is bounded above by

$$
\begin{aligned}
L_1\alpha_p\alpha_q(1-\gamma)\sum_{j,k}((1-\gamma)(1-\alpha_p))^j((1-\gamma)(1-\alpha_q))^k &= \frac{L_1\alpha_p\alpha_q(1-\gamma)}{(\gamma+\alpha_p-\gamma\alpha_p)(\gamma+\alpha_q-\gamma\alpha_q)}\\
&\le \frac{L_1\alpha_p\alpha_q(1-\gamma)}{\gamma^2};
\end{aligned}
$$

which is O ("^{2}). Thus we can ignore the second summand. Taking $L(j;k) = \frac{l\ m}{j} + \frac{l\ m}{k} \quad \frac{}{j+k+1}$, we get that

$$
\begin{aligned}
E(\#\text{-hbmsg}^{avg} \mid F_3) \quad & \sum_{j;k} \frac{\frac{l\ m}{j} + \frac{l\ m}{k}}{(j+k+1)} (1 \quad _p)^j (1 \quad _q)^k \quad _p \quad _q \\
= \quad & \sum_{j;k} \frac{1}{} (1 \quad _p)^j (1 \quad _q)^k \quad _p \quad _q \\
+ \quad & \sum_{j;k} \frac{L(j;k)}{(j+k+1)} (1 \quad _p)^j (1 \quad _q)^k \quad _p \quad _q \\
= \quad & \frac{1}{} + \frac{1}{} \sum_{j;k} \frac{L(j;k)}{j+k+1} (1 \quad _p)^j (1 \quad _q)^k \quad _p \quad _q :
\end{aligned}
$$

It clearly su ces to show that the second summand above is O ("). Note that $\frac{1}{j+k+1} < \overset{p}{\ } \overline{\ }_p$ if $j > \frac{p\,1}{p}$; similarly, $\frac{1}{j+k+1} < \overset{p}{\ } \overline{\ }_q$ if $k > \frac{p\,1}{q}$. Finally, it is clear that $\frac{1}{j+k+1} \quad 1$ for all $j;k \quad 0$. Call the second summand above S. Since $L(j;k) < 2$, we have that

$$
\begin{aligned}
S \quad & 2 \, q \overline{\ }_p \sum_{j > \frac{p\,1}{p}} \sum_{k} (1 \quad _p)^j (1 \quad _q)^k \quad _p \quad _q \\
& + 2 \, q \overline{\ }_q \sum_{j} \sum_{k > \frac{p\,1}{q}} (1 \quad _p)^j (1 \quad _q)^k \quad _p \quad _q \\
& + \sum_{j \quad \frac{p\,1}{p}} \sum_{k \quad \frac{p\,1}{q}} 2 \quad _p \quad _q \\
& 2 (p \overline{\ }_p + p \overline{\ }_q) + 2 \, p \overline{\ \quad }_{p \quad q} :
\end{aligned}
$$

Since we assumed that $p \overline{\ }_p$ and $p \overline{\ }_q$ are both O (") for this theorem, the second summand above is O ("). Thus, $E(\#\text{-hbmsg}^{avg} \mid F_3) \quad \frac{1}{}$. It follows that $E(\#\text{-hbmsg}^{avg}) \quad \frac{1}{}$, as desired. ▮

References

[ACT97] M.K. Aguilera, W. Chen, and S. Toueg. Heartbeat: a timeout-free failure detector for quiescent reliable communication. In Proceedings of the 11th International Workshop on Distributed Algorithms, pages 126{140. Springer-Verlag, Berlin/Heidelberg/New York, 1997. A full version is also available as Technical Report 97-1631, Department of Computer Science, Cornell University, 1997.

[BBS98] S. Bajaj, L. Breslau, and S. Shenker. Uniform versus priority dropping for layered video. Submitted for publication, 1998.

[BMPP98] J.-C. Bermond, N. Marlin, D. Peleg, and S. Perennes. Directed virtual path layouts in ATM networks. In Proceedings of the 12th International Symposium on Distributed Computing, Lecture Notes in Computer Science, Vol. 1499, pages 75{88, Berlin/Heidelberg/New York, 1998. Springer-Verlag.

[BS98] L. Breslau and S. Shenker. Best-e ort versus reservations: A simple comparative analysis. Submitted for publication, 1998.

[CM98] I. Cidon and O. Mokryn. Propagation and leader election in a multihop broadcast environment. In Proceedings of the 12th International Symposium on Distributed Computing, Lecture Notes in Computer Science, Vol. 1499, pages 104{118. Springer-Verlag, Berlin/Heidelberg/New York, 1998.

[EHWG98] W. Eberly, L. Higham, and J. Warpechowska-Gruca. Long-lived, fast, waitfree renaming with optimal name space and high throughput (extended abstract). In Proceedings of the 12th International Symposium on Distributed Computing, Lecture Notes in Computer Science, Vol. 1499, pages 149{160. Springer-Verlag, Berlin/Heidelberg/New York, 1998.

[FMS98] P. Flocchini, B. Mans, and N. Santoro. Sense of direction in distributed computing. In Proceedings of the 12th International Symposium on Distributed Computing, Lecture Notes in Computer Science, Vol. 1499, pages 1{15. Springer-Verlag, Berlin/Heidelberg/New York, 1998.

[Fra86] N. Francez. Fairness. Springer-Verlag, Berlin/New York, 1986.

[Kes97] S. Keshav. An Engineering Approach to Computer Networking. Professional Computing Series. Addison-Wesley, Reading, Massachusetts, 1997.

[KL95] Y. A. Korilis and A. A. Lazar. On the existence of equilibria for noncooperative ow control. Journal of the Association for Computing Machinery, 42(3):584{613, May 1995.

[LS98] A. A. Lazar and N. Semret. The PSP auction mechanism for network resource sharing. In 8th International Symposium on Dynamic Games and Applications, pages 359{365, 1998.

[MHW96] A. R. Mikler, V. Honavar, and J. S. K. Wong. Analysis of utility-theoretic heuristics for intelligent adaptive network routing. In Proceedings, Thirteenth National Conference on Arti cial Intelligence (AAAI '96), volume 1, pages 96{101, 1996.

[MIB98] J.-M. Menaud, V. Issarny, and M. Banâtre. A new protocol for e cient cooperative transversal web caching. In Proceedings of the 12th International Symposium on Distributed Computing, Lecture Notes in Computer Science, Vol. 1499, pages 288{302. Springer-Verlag, Berlin/Heidelberg/New York, 1998.

[Res87] M. D. Resnik. Choices: An Introduction to Decision Theory. University of Minnesota Press, Minneapolis, 1987.

[RS89] J. E. Russo and P. J. H. Schoemaker. Decision Traps: Ten Barriers to Brilliant Decision-Making and How to Overcome Them. Doubleday, New York, 1989.

[SKS97] A. Silberschatz, H. Korth, and S. Sudarshan. Database System Concepts. McGraw-Hill Companies, Inc., New York, third edition, 1997.

[TRAR98] F. J. Torres-Rojas, M. Ahamad, and M. Raynal. Lifetime based consistency protocols for distributed objects. In Proceedings of the 12th International Symposium on Distributed Computing, Lecture Notes in Computer Science, Vol. 1499, pages 378{392. Springer-Verlag, Berlin/Heidelberg/New York, 1998.

[YAGW 98] C. H. Young, N. B. Abu-Ghazaleh, and P. A. Wilsey. OFC: a distributed fossil-collection algorithm for time-warp. In Proceedings of the 12th International Symposium on Distributed Computing, Lecture Notes in Computer Science, Vol. 1499, pages 408{418. Springer-Verlag, Berlin/Heidelberg/New York, 1998.