A Decision-Theoretic Approach to Reliable Message Delivery*

Francis Chu and Joseph Halpern

Department of Computer Science
Upson Hall, Cornell University
Ithaca, NY 14853-7501, USA
{fcc,halpern}@cs.cornell.edu

Abstract. We argue that the tools of decision theory need to 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.

1 Introduction

In designing and running systems, decisions must always be made: When should we garbage collect? Which transactions should be aborted? 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. (See [RS89] for some common mistakes people make when making decisions.) 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. This paper can be viewed 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. Should she stay or leave? That depends. Clearly, if Bob is an important business 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,

* 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.

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 meets Bob 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 type of information we need to make this decision is precisely that considered in the Alice-Bob example above: probabilities and utilities. While it is obvious the amount of time Alice should wait depends on the situation, atomic commit protocols do not seem to take the importance of the transaction or the cost of waiting into account when deciding how long to wait. Instead, they typically have a fixed timeout period; if p_c has not heard from a process by this time, then the process is declared faulty and the transaction is aborted.

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. 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 into 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 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 (for further details, see http://www.microsoft.com/research/dtg). However, 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 first give a brief review of 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 [ACT97]. However, the heartbeat messages themselves come at a cost; this cost is investigated in Section 5. We offer some conclusions in Section 6.

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. Here, we assume that (a) we have a set $\mathcal{O}$ of possible states of the world or *outcomes*, (b) the agent can assign a real-valued *utility* to each outcomes in $\mathcal{O}$, and (c) each action or choice a of the agent can be associated with a subset $\mathcal{O}_{\mathsf{a}}$ of $\mathcal{O}$ and a probability distribution on $\mathcal{O}_{\mathsf{a}}$. 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 $\mathcal{O}_{\mathsf{a}}$ of outcomes associated with an action or choice a are the outcomes that might arise if a is performed or chosen; the probability distribution on $\mathcal{O}_{\mathsf{a}}$ 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 $\mathcal{O}$ of outcomes is a function from $\mathcal{O}$ to the reals. Given a random variable X and a probability distribution Pr on the outcomes, the expected value of X with respect to Pr, denoted $\mathbf{E}_{\Pr}(X)$, is $\sum_{o \in \mathcal{O}} \Pr(o)X(o)$. (We drop the subscript 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 the probability distribution associated with the choice. The "rational choice" is then 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. Under this assumption, the utility of the outcome (m_a, m_b) is $\mathbf{meet\text{-}Bob} + m_b\mathbf{c\text{-}wait}$ if $m_a \geq m_b$ and $m_a\mathbf{c\text{-}wait}$ if $m_a < m_b$.

Of course, in practice, the utilities might be much more complicated and need not be additive. 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 PM 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 distribution over this set of outcomes, which effectively means a probability distribution 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 (and 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 the likelihood of Bob arriving. 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 $\mathcal{S}_0$ consisting of a sender p and a receiver q connected by an unreliable bidirectional link, which drops messages with probability $0 < \gamma < 1$. If a message is not dropped, the transmission delay is τ. A process is *correct* if it never crashes. Let $x \in \{p, q\}$ and let α_x be the probability that x is correct. That is, if we select a run (uniformly) at random, the probability that x is correct in that run is α_x. In runs in which x is not correct, x crashes in each time unit with probability $\beta_x > 0$. We assume that events of p being correct or of crashing if it is not correct are both independent of the corresponding events for q.

We remark that the assumption that $\alpha_p = \alpha_q = 0$ is the one that seems most reasonable to us in practice—there is always a positive probability that a process will crash in any given round.[2] We allow the possibility that $\alpha_x \neq 0$ 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 that processes stay up forever "for all practical purposes" (for example, if the system is replaced before the process crashes).

We want to implement a reliable link on top of the unreliable one. More precisely, suppose we use send/receive to denote the (unreliable) send and receive provided by the link. We want to implement reliable high-level SEND/RECEIVE actions as a sequence of send/receive events. What specification should SEND and RECEIVE satisfy? Clearly we do not want them to create messages out of whole cloth. Thus, we certainly want the following requirement:

$\mathbf{S}_0$ If p SENDs m then p must also send m (possibly at some earlier time); if q RECEIVEs m from p, then q must also receive m from p and at some earlier time q must SEND m to q.

We shall implicitly assume $\mathbf{S}_0$ without further comment throughout the paper.

The more interesting question is what reliability requirements should we impose on SEND/RECEIVE. Perhaps the most obvious requirement is

$\mathbf{S}_1$. If p and q are correct and p SENDs m then q eventually RECEIVEs m.

Although $\mathbf{S}_1$ is very much in the spirit of typical specifications, which only focuses 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 specifications). The first shows that it is too weak: If $\alpha_p = \alpha_q = 0$, then p and q are correct (i.e., never crash) with probability 0. Thus, specification $\mathbf{S}_1$ is rather uninteresting in this case; it is saying something about a set of runs with vanishingly small likelihood. The second problem shows that $\mathbf{S}_1$ is too strong: In runs where p and q are correct, there is a chance (albeit a small one) that p sends the message m infinitely often, yet q never RECEIVEs it. In this case $\mathbf{S}_1$ is not satisfied.

[2] 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 $\mathbf{S}_1$ 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 RECEIVE the message 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:

$\mathbf{S}_2$. $\lim_{t\to\infty} \Pr(q \text{ RECEIVEs } m \text{ by time } t \mid p \text{ SENDs } m \text{ at time 0 and } p \text{ and } q \text{ do not crash up to time } t) = 1$.

Requirement $\mathbf{S}_2$ avoids the two problems we saw with $\mathbf{S}_1$. It says, in a precise sense, that if p and q are up for sufficiently long, then q will RECEIVE m (where now "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.

While we believe $\mathbf{S}_2$ is a better specification of what is desired than $\mathbf{S}_1$, it is is still not good enough for our purposes, since it does not take costs into account. For example, it is possible to satisfy $\mathbf{S}_2$ using an algorithm that has p sending its Nth message only at time 2^N. This does not seem so reasonable. As a first step to thinking in terms of costs, consider the following specification:

$\mathbf{S}_3$. The expected cost of a SEND is finite.

As stated, $\mathbf{S}_3$ is not well defined, since we have not specified the cost model. We now consider a particularly simple cost model, much in the spirit of the Alice-Bob example discussed in Section 2. Suppose we have a protocol that implements SEND and RECEIVE. 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; the utility of a run is its negative cost.

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 $n_{\mathsf{send}}(r)$ be the number of invocations of send made by the protocol during run r. If one of the processes crashes in run r before q RECEIVEs m, let $t_{wait}(r)$ be the time of the crash. Otherwise, if q eventually RECEIVEs m in r, let $t_{wait}(r)$ be the time q RECEIVEs m. Let $t_{wait}(r) = \infty$ if both processes are correct and q never RECEIVEs m in r. The cost of waiting in run r is then $t_{wait}(r)$**c-wait**. (Implicitly, we assume here that p starts out wanting to SEND m to q at time 0.) Intuitively, $t_{wait}(r)$ measures the amount of time q has to wait until it RECEIVEs m. There

is a subtle issue of how to define $t_{wait}(r)$ if one of the processes crashes before q RECEIVEs m. If q crashes first, then it seems reasonable to stop penalizing the protocol at this point; after all, after q has crashed, it is no longer waiting for the message. It is not so obvious what to do if p crashes first. For definiteness, we have taken an approach that is consistent with specifications like $\mathbf{S}_1$ and have decided that the protocol has no further obligation when p crashes; we capture this by not charging for waiting time from that point on. While this choice suffices for our purposes, it is clearly not the only reasonable choice that could have been made. This just emphasizes some of the subtleties involved in defining a cost model.

We take the cost of run r to be $\mathbf{c}\text{-SEND}(r) = n_{\mathsf{send}}(r)\mathbf{c}\text{-send}+t_{wait}(r)\mathbf{c}\text{-wait}$. Note that $\mathbf{c}$-SEND is a random variable on runs; $\mathbf{S}_3$ says that we want $\mathbf{E}(\mathbf{c}\text{-SEND})$ to be finite.

In this cost model, $\mathbf{S}_2$ and $\mathbf{S}_3$ are incomparable. Consider a protocol where p sends m to q in every round until it receives an ACK message, and q sends its kth ACK message N^k rounds after receiving m for the kth time, where $N\gamma > 1$. It is easy to see that this protocol satisfies $\mathbf{S}_2$ but not $\mathbf{S}_3$. On the other hand, suppose $\alpha_p = \alpha_q = 0$. Then it is easy to see that the trivial protocol (where no messages are sent at all) satisfies $\mathbf{S}_3$. One of p or q will crash with probability $\beta_p + \beta_q - \beta_p\beta_q$, so one of them is expected to crash by time $1/(\beta_p + \beta_q - \beta_p\beta_q)$. Thus, $\mathbf{E}(\mathbf{c}\text{-SEND}) = \mathbf{c}\text{-wait}/(\beta_p + \beta_q - \beta_p\beta_q)$ for the trivial protocol, so the trivial protocol satisfies $\mathbf{S}_3$, although it clearly does not satisfy $\mathbf{S}_2$.

The following theorem characterizes when $\mathbf{S}_3$ is implementable, using the cost model above. Moreover, it shows that with this cost model, when $\mathbf{S}_3$ is satisfiable, there are in fact protocols that satisfy $\mathbf{S}_3$ and $\mathbf{S}_2$ simultaneously.

Theorem 1. *If* $\mathbf{c}$-SEND *characterizes the cost of a run, then* $\mathbf{S}_3$ *is implementable in* $\mathcal{S}_0$ *iff* $\alpha_p = 0$ *or* $\alpha_q = 0$ *or* $\alpha_q = 1$ *or* $\alpha_p = 1$. *Moreover, if* $\alpha_p = 0$ *or* $\alpha_q = 0$ *or* $\alpha_q = 1$ *or* $\alpha_p = 1$, *then there is a protocol that satisfies both* $\mathbf{S}_2$ *and* $\mathbf{S}_3$.

Proof: Suppose $\alpha_q = 1$ or $\alpha_p = 0$. Consider the protocol in which p SENDs m by sending m to q at every round it gets ACK(m) from q and q sends ACK(m) whenever it receives m. Thus, the SEND starts when p first sends m and ends when p receives the first ACK(m) from q; the RECEIVE starts and ends the first time q receives m. To see that this works, first consider the case that $\alpha_q = 1$. Let C_p be the set of runs in which p receives an ACK(m) from q before crashing. Clearly the probability that p sends a message m which is received by q is $1-\gamma$; the probability that both m is received by q and the corresponding ACK(m) is received by p is $(1-\gamma)^2$. Suppose the N_1th copy of m is the one first received by q and the N_2th copy is the one whose acknowledgement is first received by p. We expect N_1 to be $\frac{1}{(1-\gamma)}$ in runs of C_p and N_2 to be $\frac{1}{(1-\gamma)^2}$. Since a message takes τ to arrive and messages are sent in every round, it follows that $\mathbf{E}(t_{wait}|C_p) = \frac{1}{(1-\gamma)}+\tau$. Moreover, since p stops sending when it receives ACK(m) from q, it will stop 2τ rounds after sending the N_2th copy of m. Thus, p expects to send $\frac{1}{(1-\gamma)^2} + 2\tau - 1$ messages in runs of C_p. We expect q to receive $1-\gamma$ of these messages and then send an ACK(m) in response to each of them. Thus, q

expects to send $\frac{1}{(1-\gamma)} + (2\tau - 1)(1-\gamma)$ messages in runs of C_p. We conclude that $\mathbf{E}(n_{\mathsf{send}}|C_p) = \frac{1}{(1-\gamma)} + \frac{1}{(1-\gamma)^2} + (2\tau - 1)(2-\gamma)$.

It is easy to see that $\mathbf{E}(t_{wait}|\overline{C_p}) \leq \mathbf{E}(t_{wait}|C_p)$, since in runs of $\overline{C_p}$, p crashes before it receives an ACK(m) from q. Similarly, $\mathbf{E}(n_{\mathsf{send}}|\overline{C_p}) \leq \mathbf{E}(n_{\mathsf{send}}|C_p)$. It follows that $\mathbf{E}(\mathbf{c}\text{-}\mathsf{SEND}) \leq (\frac{1}{(1-\gamma)} + \frac{1}{(1-\gamma)^2} + (2\tau-1)(2-\gamma))\mathbf{c}\text{-send} + (\frac{1}{(1-\gamma)} + \tau)\mathbf{c}\text{-wait}$. Thus, this protocol satisfies $\mathbf{S}_3$.

For $\mathbf{S}_2$, note that for $t \geq \tau$, the probability that q does not RECEIVE m by time t given that both p and q are still up, is $\gamma^{t-\tau}$. Thus $\mathbf{S}_2$ is also satisfied.

Now consider the case that $\alpha_p = 0$. Note that in this case, p is expected to crash at time $1/\beta_p$. Thus, $\mathbf{E}(t_{wait}) \leq 1/\beta_p$ and $\mathbf{E}(n_{\mathsf{send}}) \leq 2/\beta_p$ (since each time unit p is up it sends m exactly once and q will send at most one ACK(m) in response), regardless of whether q is correct. Thus $\mathbf{E}(\mathbf{c}\text{-}\mathsf{SEND})$ is again finite. The argument that $\mathbf{S}_2$ is satisfied is the same as before.

Now suppose $\alpha_p = 1$ or $\alpha_q = 0$. These cases are somewhat analogous to the ones above, except we need a "receiver-driven" protocol. Consider a protocol in which q continually queries p until it gets a message from p. More precisely, if we want to SEND/RECEIVE a number of messages, we assume that each query is numbered. The kth SEND starts when p gets the first query numbered k from q and ends with the send event just prior to when p receives the first query numbered $k+1$ from q. (Note that this means that p does not know that the kth SEND has ended until the $(k+1)$th begins.) The kth RECEIVE starts when q sends the first query numbered k to p and ends when q receives the kth message from p. By reasoning similar to the previous cases, we can show that $\mathbf{E}(n_{\mathsf{send}})$ and $\mathbf{E}(t_{wait})$ are both finite and that $\mathbf{S}_2$ is satisfied.

We now turn to the negative result. We show that $\mathbf{c}\text{-}\mathsf{SEND} = \infty$ with some probability $\varepsilon > 0$. (Note that this is stronger than we need.) Suppose $0 < \alpha_p < 1$ and $0 < \alpha_q < 1$. Let P be a protocol. Let $R_1 = \{r : q$ crashes at time 0 and p is correct in $r\}$. Note that p will do the same thing in all runs in R_1. Either p stops sending messages after some time t or p never stops sending messages. If p never stops, then $n_{\mathsf{send}} = \infty$ for each $r \in R_1$ and we can take ε to be $\alpha_p(1-\alpha_q)\beta_q$, the probability of R_1. Now suppose p stops after time t. Let $R_2 = \{r : p$ crashes at time 0 and q is correct in $r\}$. Note that q will do the same thing in all runs of R_2. Either q stops sending messages after some time t' or q never stops sending messages. If q never stops, the same reasoning as above works. So suppose q stops after time t'. Let $t'' \geq t, t'$. Consider $R_3 = \{r :$ both processes are correct and all messages up to time t'' are lost in $r\}$. Let n_p be the number of messages sent by p and n_q be the number of messages sent by q. Then $t_{wait} = \infty$ for all $r \in R_3$ and we can take ε to be $\alpha_p\alpha_q\gamma^{n_p+n_q}$. ■

Of course, once we think in terms of utility-based specifications like $\mathbf{S}_3$, we do not want to know just whether a protocol implements $\mathbf{S}_3$. We are in a position to compare the performance of different protocols that implement $\mathbf{S}_3$ (or of variants of one protocol that all implement $\mathbf{S}_3$) by considering their expected utility. Let P_s^δ and P_r^δ be generalizations (in the sense that they send messages every δ rounds where δ need not be 1) of the sender-driven and receiver-driven protocols from

Theorem 1, respectively. Let P_{tr} denote the trivial (i.e., "do nothing") protocol. We use $\mathbf{E}^P$ to denote the expectation operator determined by the probability measure on runs induced by using protocol P. Thus, for example, $\mathbf{E}^{\mathrm{P}^\delta_s}(n_{\mathsf{send}})$ is the expected number of messages sent by P^δ_s. If $\alpha_p = \alpha_q = 0$, then P^δ_s, P^δ_r, and P_{tr} all satisfy $\mathbf{S}_3$ (although P_{tr} does not satisfy $\mathbf{S}_2$). Which is better?

In practice, process failures and link failures are very unlikely events. So we assume, for the rest of the paper, that β_p, β_q, and γ are all very small, so that we can ignore sums of products of these terms (with coefficients like $2\tau^2$, δ, etc.).[3] (Informally, products involving α_p, α_q, and γ are $O(\varepsilon)$ terms and $2\tau^2$, δ, etc., are $O(1)$ terms.) We write $t_1 \approx t_2$ if t_1 and t_2 differ by an expression that is the sum of products involving (at least) one of β_p, β_q or γ. Note that we do not assume expressions like β_p/β_q and β_q/β_p are small.

Proposition 2. *If $\alpha_p = \alpha_q = 0$, then*

$\mathbf{E}^{\mathrm{P}_{tr}}(t_{wait}) \approx \frac{1}{\beta_p+\beta_q}$	$\mathbf{E}^{\mathrm{P}^\delta_s}(t_{wait}) \approx \tau$	$\mathbf{E}^{\mathrm{P}^\delta_r}(t_{wait}) \approx 2\tau$
$\mathbf{E}^{\mathrm{P}_{tr}}(n_{\mathsf{send}}) = 0$	$\mathbf{E}^{\mathrm{P}^\delta_s}(n_{\mathsf{send}}) \approx \frac{\tau\beta_q}{\delta\beta_p} + 2\left\lceil\frac{2\tau}{\delta}\right\rceil$	$\mathbf{E}^{\mathrm{P}^\delta_r}(n_{\mathsf{send}}) \approx \frac{\tau\beta_p}{\delta\beta_q} + 2\left\lceil\frac{2\tau}{\delta}\right\rceil$.

Proof: See the full paper. ■

Note that the expected cost of messages for P^δ_s is the same as that for P^δ_r, except that the roles of β_p and β_q are reversed. The expected time cost of P^δ_r is roughly τ higher than that of P^δ_s, because q cannot RECEIVE m before time 2τ with a receiver-driven protocol, whereas q may RECEIVE m as early as τ 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 δ very large to minimize costs. (Intuitively, the larger δ, 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 SEND/RECEIVE in isolation, As we shall see in Section 5, this conclusion is no longer justified once we consider repeated invocations of SENDs/RECEIVEs. Finally, note that if the cost of messages is high, waiting is cheap, and q is quite likely to fail (relative to p), then p is better off (according to this cost measure) not sending any messages at all (i.e., using P_{tr}) than sending them in the hope of getting a response for q.

Thus, as far as $\mathbf{S}_3$ is concerned, there are times when P_{tr} is better than P^δ_s or P^δ_r. How much of a problem is it that P_{tr} does not satisfy $\mathbf{S}_2$? Our claim is that if this desideratum (i.e., $\mathbf{S}_2$) is important, then it should be reflected in the cost measure. While the cost measure we have chosen does does take into account waiting time, it does not penalize it sufficiently to give us $\mathbf{S}_2$. It is not too hard to find a cost measure that captures $\mathbf{S}_2$. For example, suppose we take **c-**$\mathsf{SEND}^*(r) = N^{t_{wait}}$, where $N(1 - \beta_p - \beta_q + \beta_p\beta_q) > 1$.[4]

[3] Of course, with a little more effort, we can do more exact calculations, but doing this yields no further insight and results in unwieldy expressions.

[4] Note that $\mathbf{S}_3$ may be unsatisfiable for certain values of N.

Proposition 3. *If* **c-**SEND* *characterizes the cost of a run, then* $\mathbf{S}_3$ *implies* $\mathbf{S}_2$.

Proof: See the full paper. ■

The moral here is that $\mathbf{S}_3$ gives us the flexibility to specify what really matters in a protocol, by appropriately describing the cost measure.

4 Using Heartbeats

We saw in Section 3 that $\mathbf{S}_3$ 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). Aguilera, Chen, and Toueg [ACT97] (ACT from now on) suggest an approach that avoids this problem, using *heartbeat* messages.[5] 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 processes may not crash and communication is unreliable, a result much in the spirit of the negative part of Theorem 1.[6] In this section, we show that (using the linear cost model of the previous section) we can use heartbeats to implement $\mathbf{S}_3$ 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 RC 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) HBMSG has arrived; if so p sends m to q, provided it has not already received an ACK from q; q sends ACK(m) every time it receives m. Note that according to this protocol, q does not send any HBMSGs. 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 mechanism. By using the cost measure of the previous section, we actually do not count the cost of HBMSGs. It is also worth noting that this is a sender-driven protocol, quite like that given in the proof of Theorem 1. It is straightforward to also design a receiver-driven protocol using heartbeats.

[5] The heartbeat messages are used to implement what they call a *heartbeat failure detector*, which is a distributed oracle.

[6] 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 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:

```
1. while ¬receive(ACK(m)) do
2.    if receive(HBMSG) then
3.       send(m)
4.    fi
5. od
```

The receiver's protocol:

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

Fig. 1. Protocol RC

We now want to show that RC implements $\mathbf{S}_3$ and get a good estimate of the actual cost.

Theorem 4. *If* **c-**SEND *characterizes the cost of a run, then Protocol* RC *implements* $\mathbf{S}_3$ *in* $\mathcal{S}_0$. *Moreover,* $\mathbf{E}(t_{wait}) \approx 2\tau$ *and* $\mathbf{E}(n_{\mathsf{send}}) \approx 2\left\lceil \frac{2\tau}{\delta} \right\rceil$, *so that* $\mathbf{E}(\textbf{c-}\mathsf{SEND}) \approx 2\tau\textbf{c-wait} + 2\left\lceil \frac{2\tau}{\delta} \right\rceil \textbf{c-}\mathsf{send}$.

Proof: See the full paper. ■

The analysis of RC is much like that of P_s^δ in Proposition 2. Indeed, in the case that $\alpha_p = \alpha_q = 0$, the two protocols are almost identical. The waiting time is roughly τ more in RC, since p does not start sending until it receives the first HBMSG from q. On the other hand, we are better off using RC if q crashes before acknowledging p's message. In this case, with P_s^δ, p continues to send until it crashes, while with RC, 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 α_p and α_q are between 0 and 1, we need heartbeats or something like them to get around the impossibility result of Theorem 1. But if $\alpha_p = \alpha_q = 0$, 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.

5 The Cost of Heartbeats

In the last section we showed that $\mathbf{S}_3$ is achievable with the help of heartbeats. When we computed the expected costs, however, we were not counting the HBMSGs because the heartbeats were not sent by the protocol. This cost model is appropriate if we consider the heartbeat layer as given, and are just interesting in designing a protocol that takes advantage of it. But it is not appropriate if we are trying to decide whether to use a heartbeat protocol at all or how frequently heartbeats should be sent.

As evidence of this, note that it is immediate from Theorem 4 that under the cost model we have been using, the choice of δ that minimizes the expected

cost is clearly at most $2\tau + 1$. 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 RC has completed.

The whole point of heartbeat messages is that they are meant to be used, not just by one invocation of a 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 RC protocol is invoked from time to time, by different 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 δ 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 RC. (It is possible to do this with n processes, but the two-process case suffices to illustrate the main point, which is that the optimal δ should depend on how often RC is invoked.) We assume that each process, while it is running, invokes RC with probability σ at each time unit. Thus, informally, at every round, each running process tosses a coin with probability of σ of landing heads. If it lands heads, the process then invokes RC with the other as the recipient.

Roughly speaking, in computing the cost of a run, we consider the cost of each invocation of RC together with the cost of all the heartbeat messages sent in the run. Our interest will then be in the cost *per invocation of* RC. Thus, we apportion the cost of the heartbeat messages among the invocations of RC. If there are relatively few invocations of RC, 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 RC is invoked, a different message is sent. (For example, messages could be numbered and include the name of the sender and recipient.) We say RC(m) *is invoked at time* t_1 *in* r if at time t_1 some process x first executes line 1 of the code of the sender with message m. This invocation of RC *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 the receiver crashed before x invokes RC(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 x stops sending m because it received ACK(m). Thus, with probability 1, all invocations of RC will eventually complete. (Note that we are still not counting heartbeat messages.) Let $\mathbf{cmp}(r, T)$ be the number of invocations of RC that have completed by time T in r; let $\mathbf{c\text{-}SEND}(r, T)$ be the cost of these

invocations. Let **c-hb**(r,T) be the cost of the HBMSGs up to time T in r. This is simply the number of HBMSGs sent up to time T (which we denote by **n-hb**(r,T)) multiplied by **c-send**. Let **t-cst**$(r,T) =$ **c-SEND**$(r,T) +$ **c-hb**(r,T). Finally, let **c-SEND**$'(r) = \limsup_{T\to\infty}$ **t-cst**$(r,T)/($**cmp**$(r,T)+1)$, where lim sup denotes the limit of the supremum, that is, $\lim_{T'\to\infty}\sup_{0\le T\le T'}$ **t-cst**$(r,T)/($**cmp**$(r,T)+1)$.[7] Thus **c-SEND**$'(r)$ is essentially the average cost per invocation of RC, 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, we assume that $\sqrt{\beta_p}$ and $\sqrt{\beta_q}$ are also $O(\varepsilon)$.

Theorem 5. **E**(**c-SEND**$') \approx ((1-\alpha_p)(1-\alpha_q)\lambda + \alpha_p\alpha_q)\left(2\left\lceil\frac{2\tau}{\delta}\right\rceil\textbf{c-send} + (2\tau + \frac{\delta}{2})\textbf{c-wait}\right) + \frac{1}{\delta\sigma}$**c-send**, *where* $0<\lambda<1$ *is a constant that depends only on* β_p *and* β_q.[8]

Proof: See the full paper. ■

Note that with this cost model, we have a real decision to make in terms of how frequently to send heartbeats. As before, there is some benefit to making $\delta > 2\tau$, to minimize the number of useless messages sent when RC is invoked (that is, messages sent by the sender before receiving the receiver's acknowledgement). Also, by making δ larger we will send fewer heartbeat messages between invocations of RC. On the other hand, if we make δ too large, then the sender may have to wait a long time after invoking RC 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 δ. Clearly we can find an optimal choice for δ by standard calculus.

In the model just presented, if **c-wait** is large enough relative to **c-send**, we will take δ to be 1. Taking δ 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) δ should have, we need to carefully look at the actual system and the cost model.

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

[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.

[8] Intuitively, λ depends on the probability that both p and q are up relative to the probability that one is up and the other down.

the decision depends on the cost model. None of our results are deep; the cost model 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 exactly what the cost model is and encourages the elicitation of utilities from users. We believe 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 .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.

Clearly another important direction for research is to consider *qualitative decision theory*, where both our measures of utility and probability and more qualitative, and not necessarily real numbers. This is, in fact, an active area of current research, as the bibliography of 290 papers at http://walrus.stanford.edu/diglib/csbibliography/Ai/qualitative.decision.theory.html 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 specification, we hope that this discussion has convinced the reader of the promise of the approach.

Acknowledgements

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.

References

[ACT97] M. K. Aguilera, W. Chen, and S. Toueg. Heartbeat: a timeout-free failure detector for quiescent reliable communication. In *Proc. 11th Int. Workshop on Distributed Algorithms*, pages 126–140. Springer-Verlag, 1997. A full version is available as Cornell University Technical Report 97-1631, 1997.

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

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

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

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

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

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

This article was processed using the LaTeX macro package with LLNCS style