The Part-Time Parliament

Leslie Lamport

This article appeared in *ACM Transactions on Computer Systems 16*, 2 (May 1998), 133-169. Minor corrections were made on 29 August 2000.

The Part-Time Parliament

LESLIE LAMPORT
Digital Equipment Corporation

Recent archaeological discoveries on the island of Paxos reveal that the parliament functioned despite the peripatetic propensity of its part-time legislators. The legislators maintained consistent copies of the parliamentary record, despite their frequent forays from the chamber and the forgetfulness of their messengers. The Paxon parliament's protocol provides a new way of implementing the state-machine approach to the design of distributed systems.

Categories and Subject Descriptors: C2.4 [**Computer-Communications Networks**]: Distributed Systems—*Network operating systems*; D4.5 [**Operating Systems**]: Reliability—*Fault-tolerance*; J.1 [**Administrative Data Processing**]: Government

General Terms: Design, Reliability

Additional Key Words and Phrases: State machines, three-phase commit, voting

This submission was recently discovered behind a filing cabinet in the *TOCS* editorial office. Despite its age, the editor-in-chief felt that it was worth publishing. Because the author is currently doing field work in the Greek isles and cannot be reached, I was asked to prepare it for publication.

The author appears to be an archeologist with only a passing interest in computer science. This is unfortunate; even though the obscure ancient Paxon civilization he describes is of little interest to most computer scientists, its legislative system is an excellent model for how to implement a distributed computer system in an asynchronous environment. Indeed, some of the refinements the Paxons made to their protocol appear to be unknown in the systems literature.

The author does give a brief discussion of the Paxon Parliament's relevance to distributed computing in Section 4. Computer scientists will probably want to read that section first. Even before that, they might want to read the explanation of the algorithm for computer scientists by Lampson [1996]. The algorithm is also described more formally by De Prisco et al. [1997]. I have added further comments on the relation between the ancient protocols and more recent work at the end of Section 4.

Keith Marzullo
University of California, San Diego

Authors' address: Systems Research Center, Digital Equipment Corporation, 130 Lytton Avenue, Palo Alto, CA 94301.
Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.
© 1998 ACM 0000-0000/98/0000-0000 $00.00

1 The Problem

1.1 The Island of Paxos

Early in this millennium, the Aegean island of Paxos was a thriving mercantile center.[1] Wealth led to political sophistication, and the Paxons replaced their ancient theocracy with a parliamentary form of government. But trade came before civic duty, and no one in Paxos was willing to devote his life to Parliament. The Paxon Parliament had to function even though legislators continually wandered in and out of the parliamentary Chamber.

The problem of governing with a part-time parliament bears a remarkable correspondence to the problem faced by today's fault-tolerant distributed systems, where legislators correspond to processes and leaving the Chamber corresponds to failing. The Paxons' solution may therefore be of some interest to computer scientists. I present here a short history of the Paxos Parliament's protocol, followed by an even shorter discussion of its relevance for distributed systems.

Paxon civilization was destroyed by a foreign invasion, and archeologists have just recently begun to unearth its history. Our knowledge of the Paxon Parliament is therefore fragmentary. Although the basic protocols are known, we are ignorant of many details. Where such details are of interest, I will take the liberty of speculating on what the Paxons might have done.

1.2 Requirements

Parliament's primary task was to determine the law of the land, which was defined by the sequence of decrees it passed. A modern parliament will employ a secretary to record its actions, but no one in Paxos was willing to remain in the Chamber throughout the session to act as secretary. Instead, each Paxon legislator maintained a *ledger* in which he recorded the numbered sequence of decrees that were passed. For example, legislator $\Lambda\breve{\iota}\nu\chi\partial$'s ledger had the entry

155: *The olive tax is 3 drachmas per ton*

if she believed that the 155^{th} decree passed by Parliament set the tax on olives to 3 drachmas per ton. Ledgers were written with indelible ink, and their entries could not be changed.

The first requirement of the parliamentary protocol was the *consistency of ledgers*, meaning that no two ledgers could contain contradictory information. If legislator $\Phi\iota\sigma\partial\epsilon\rho$ had the entry

132: *Lamps must use only olive oil*

in his ledger, then no other legislator's ledger could have a different entry for decree 132. However, another legislator might have no entry in his ledger for decree 132 if he hadn't yet learned that the decree had been passed.

Consistency of ledgers was not sufficient, since it could be trivially fulfilled by leaving all ledgers blank. Some requirement was needed to guarantee that decrees

[1]It should not be confused with the Ionian island of Paxoi, whose name is sometimes corrupted to *Paxos*.

were eventually passed and recorded in ledgers. In modern parliaments, the passing of decrees is hindered by disagreement among legislators. This was not the case in Paxos, where an atmosphere of mutual trust prevailed. Paxon legislators were willing to pass any decree that was proposed. However, their peripatetic propensity posed a problem. Consistency would be lost if one group of legislators passed the decree

37: *Painting on temple walls is forbidden*

and then left for a banquet, whereupon a different group of legislators entered the Chamber and, knowing nothing about what had just happened, passed the conflicting decree

37: *Freedom of artistic expression is guaranteed*

Progress could not be guaranteed unless enough legislators stayed in the Chamber for a long enough time. Because Paxon legislators were unwilling to curtail their outside activities, it was impossible to ensure that any decree would ever be passed. However, legislators were willing to guarantee that, while in the Chamber, they and their aides would act promptly on all parliamentary matters. This guarantee allowed the Paxons to devise a parliamentary protocol satisfying the following *progress condition*.

> If a majority of the legislators[2] were in the Chamber and no one entered or left the Chamber for a sufficiently long period of time then any decree proposed by a legislator in the Chamber would be passed, and every decree that had been passed would appear in the ledger of every legislator in the Chamber.

1.3 Assumptions

The requirements of the parliamentary protocol could be achieved only by providing the legislators with the necessary resources. Each legislator received a sturdy ledger in which to record the decrees, a pen, and a supply of indelible ink. Legislators might forget what they had been doing if they left the Chamber,[3] so they would write notes in the back of the ledgers to remind themselves of important parliamentary tasks. An entry in the list of decrees was never changed, but notes could be crossed out. Achieving the progress condition required that legislators be able to measure the passage of time, so they were given simple hourglass timers.

Legislators carried their ledgers at all times, and could always read the list of decrees and any note that had not been crossed out. The ledgers were made of the finest parchment and were used for only the most important notes. A legislator would write other notes on a slip of paper, which he might (or might not) lose if he left the Chamber.

The acoustics of the Chamber were poor, making oratory impossible. Legislators could communicate only by messenger, and were provided with funds to hire as

[2] In translating the progress condition, I have rendered the Paxon word $\mu\alpha\delta\zeta\partial\omega\rho\iota\breve{\pi}\sigma\epsilon\tau$ as *majority of the legislators*. Alternative translations of this word have been proposed and are discussed in Section 2.2.

[3] In one tragic incident, legislator $T\omega\upsilon\epsilon\gamma$ developed irreversible amnesia after being hit on the head by a falling statue just outside the Chamber.

many messengers as they needed. A messenger could be counted on not to garble messages, but he might forget that he had already delivered a message, and deliver it again. Like the legislators they served, messengers devoted only part of their time to parliamentary duties. A messenger might leave the Chamber to conduct some business—perhaps taking a six-month voyage—before delivering a message. He might even leave forever, in which case the message would never be delivered.

Although legislators and messengers could enter and leave at any time, when inside the Chamber they devoted themselves to the business of Parliament. While they remained in the Chamber, messengers delivered messages in a timely fashion and legislators reacted promptly to any messages they received.

The official records of Paxos claim that legislators and messengers were scrupulously honest and strictly obeyed parliamentary protocol. Most scholars discount this as propaganda, intended to portray Paxos as morally superior to its eastern neighbors. Dishonesty, although rare, undoubtedly did occur. However, because it was never mentioned in official documents, we have little knowledge of how Parliament coped with dishonest legislators or messengers. What evidence has been uncovered is discussed in Section 3.3.5.

2 The Single-Decree Synod

The Paxon Parliament evolved from an earlier ceremonial Synod of priests that was convened every 19 years to choose a single, symbolic decree. For centuries, the Synod had chosen the decree by a conventional procedure that required all priests to be present. But as commerce flourished, priests began wandering in and out of the Chamber while the Synod was in progress. Finally, the old protocol failed, and a Synod ended with no decree chosen. To prevent a repetition of this theological disaster, Paxon religious leaders asked mathematicians to formulate a protocol for choosing the Synod's decree. The protocol's requirements and assumptions were essentially the same as those of the later Parliament except that instead of containing a sequence of decrees, a ledger would have at most one decree. The resulting Synod protocol is described here; the Parliamentary protocol is described in Section 3.

Mathematicians derived the Synod protocol in a series of steps. First, they proved results showing that a protocol satisfying certain constraints would guarantee consistency and allow progress. A *preliminary protocol* was then derived directly from these constraints. A restricted version of the preliminary protocol provided the *basic protocol* that guaranteed consistency, but not progress. The complete Synod protocol, satisfying the consistency and progress requirements, was obtained by restricting the basic protocol.[4]

The mathematical results are described in Section 2.1, and the protocols are described informally in Sections 2.2–2.4. A more formal description and correctness proof of the basic protocol appears in the appendix.

---

[4]The complete history of the Synod protocol's discovery is not known. Like modern computer scientists, Paxon mathematicians would describe elegant, logical derivations that bore no resemblance to how the algorithms were actually derived. However, it is known that the mathematical results (Theorems 1 and 2 of Section 2.1) really did precede the protocol. They were discovered when mathematicians, in response to the request for a protocol, were attempting to prove that a satisfactory protocol was impossible.

2.1 Mathematical Results

The Synod's decree was chosen through a series of numbered *ballots*, where a ballot was a referendum on a single decree. In each ballot, a priest had the choice only of voting for the decree or not voting.[5] Associated with a ballot was a set of priests called a *quorum*. A ballot succeeded iff (if and only if) every priest in the quorum voted for the decree. Formally, a ballot B consisted of the following four components. (Unless otherwise qualified, *set* is taken to mean *finite set*.[6])

B_{dec} A decree (the one being voted on).

B_{qrm} A nonempty set of priests (the ballot's quorum).

B_{vot} A set of priests (the ones who cast votes for the decree).[7]

B_{bal} A ballot number.

A ballot B was said to be *successful* iff $B_{qrm} \subseteq B_{vot}$, so a successful ballot was one in which every quorum member voted.

Ballot numbers were chosen from an unbounded ordered set of numbers. If $B'_{bal} > B_{bal}$, then ballot B' was said to be *later* than ballot B. However, this indicated nothing about the order in which ballots were conducted; a later ballot could actually have taken place before an earlier one.

Paxon mathematicians defined three conditions on a set $\mathcal{B}$ of ballots, and then showed that consistency was guaranteed and progress was possible if the set of ballots that had taken place satisfied those conditions. The first two conditions were simple; they can be stated informally as follows.

$B1(\mathcal{B})$ Each ballot in $\mathcal{B}$ has a unique ballot number.

$B2(\mathcal{B})$ The quorums of any two ballots in $\mathcal{B}$ have at least one priest in common.

The third condition was more complicated. One Paxon manuscript contained the following, rather confusing, statement of it.

$B3(\mathcal{B})$ For every ballot B in $\mathcal{B}$, if any priest in B's quorum voted in an earlier ballot in $\mathcal{B}$, then the decree of B equals the decree of the latest of those earlier ballots.

Interpretation of this cryptic text was aided by the manuscript pictured in Figure 1, which illustrates condition $B3(\mathcal{B})$ with a set $\mathcal{B}$ of five ballots for a Synod consisting of the five priests A, B, Γ, Δ, and E. This set $\mathcal{B}$ contains five ballots, where for each ballot, the set of voters is the subset of the priests in the quorum whose names are enclosed in boxes. For example, ballot number 14 has decree α, a quorum containing three priests, and a set of two voters. Condition $B3(\mathcal{B})$ has the form "for every B in $\mathcal{B}$: ...", where "..." is a condition on ballot B. The conditions for the five ballots B of Figure 1 are as follows.

[5] Like some modern nations, Paxos had not fully grasped the nature of Athenian democracy.

[6] Although Paxon mathematicians were remarkably advanced for their time, they obviously had no knowledge of set theory. I have taken the liberty of translating the Paxon's more primitive notation into the language of modern set theory.

[7] Only priests in the quorum actually voted, but Paxon mathematicians found it easier to convince people that the protocol was correct if, in their proof, they allowed any priest to vote in any ballot.

#	*decree*	*quorum and* $\boxed{\textit{voters}}$				
2	α	A	B	Γ	$\boxed{\Delta}$	
5	β	A	B	$\boxed{\Gamma}$		E
14	α		$\boxed{\text{B}}$		Δ	$\boxed{\text{E}}$
27	β	$\boxed{\text{A}}$		$\boxed{\Gamma}$	$\boxed{\Delta}$	
29	β		$\boxed{\text{B}}$	Γ	Δ	

Fig. 1. Paxon manuscript showing a set $\mathcal{B}$, consisting of five ballots, that satisfies conditions $B1(\mathcal{B})$–$B3(\mathcal{B})$. (Explanatory column headings have been added.)

2. Ballot number 2 is the earliest ballot, so the condition on that ballot is trivially true.
5. None of ballot 5's four quorum members voted in an earlier ballot, so the condition on ballot 5 is also trivially true.
14. The only member of ballot 14's quorum to vote in an earlier ballot is Δ, who voted in ballot number 2, so the condition requires that ballot 14's decree must equal ballot 2's decree.
27. (This is a successful ballot.) The members of ballot 27's quorum are A, Γ, and Δ. Priest A did not vote in an earlier ballot, the only earlier ballot Γ voted in was ballot 5, and the only earlier ballot Δ voted in was ballot 2. The latest of these two earlier ballots is ballot 5, so the condition requires that ballot 27's decree must equal ballot 5's decree.
29. The members of ballot 29's quorum are B, Γ, and Δ. The only earlier ballot that B voted in was number 14, priest Γ voted in ballots 5 and 27, and Δ voted in ballots 2 and 27. The latest of these four earlier ballots is number 27, so the condition requires that ballot 29's decree must equal ballot 27's decree.

To state $B1(\mathcal{B})$–$B3(\mathcal{B})$ formally requires some more notation. A *vote* v was defined to be a quantity consisting of three components: a priest v_{pst}, a ballot number v_{bal}, and a decree v_{dec}. It represents a vote cast by priest v_{pst} for decree v_{dec} in ballot number v_{bal}. The Paxons also defined *null* votes to be votes v with $v_{bal} = -\infty$ and $v_{dec} = \text{BLANK}$, where $-\infty < b < \infty$ for any ballot number b, and BLANK is not a decree. For any priest p, they defined $null_p$ to be the unique null vote v with $v_{pst} = p$.

Paxon mathematicians defined a total ordering on the set of all votes, but part of the manuscript containing the definition has been lost. The remaining fragment indicates that, for any votes v and v', if $v_{bal} < v'_{bal}$ then $v < v'$. It is not known how the relative order of v and v' was defined if $v_{bal} = v'_{bal}$.

For any set $\mathcal{B}$ of ballots, the set $Votes(\mathcal{B})$ of votes in $\mathcal{B}$ was defined to consist of all votes v such that $v_{pst} \in B_{vot}$, $v_{bal} = B_{bal}$, and $v_{dec} = B_{dec}$ for some $B \in \mathcal{B}$. If p is a priest and b is either a ballot number or $\pm\infty$, then $MaxVote(b, p, \mathcal{B})$ was defined to be the largest vote v in $Votes(\mathcal{B})$ cast by p with $v_{bal} < b$, or to be $null_p$ if there was no such vote. Since $null_p$ is smaller than any real vote cast by p, this means that $MaxVote(b, p, \mathcal{B})$ is the largest vote in the set

$$\{v \in Votes(\mathcal{B}) : (v_{pst} = p) \wedge (v_{bal} < b)\} \cup \{null_p\}$$

For any nonempty set Q of priests, $MaxVote(b, Q, \mathcal{B})$ was defined to equal the maximum of all votes $MaxVote(b, p, \mathcal{B})$ with p in Q.

Conditions $B1(\mathcal{B})$–$B3(\mathcal{B})$ are stated formally as follows.[8]

$$\begin{aligned}
B1(\mathcal{B}) &\triangleq \forall B, B' \in \mathcal{B} : (B \neq B') \Rightarrow (B_{bal} \neq B'_{bal}) \\
B2(\mathcal{B}) &\triangleq \forall B, B' \in \mathcal{B} : B_{qrm} \cap B'_{qrm} \neq \emptyset \\
B3(\mathcal{B}) &\triangleq \forall B \in \mathcal{B} : (MaxVote(B_{bal}, B_{qrm}, \mathcal{B})_{bal} \neq -\infty) \Rightarrow \\
&\qquad\qquad (B_{dec} = MaxVote(B_{bal}, B_{qrm}, \mathcal{B})_{dec})
\end{aligned}$$

Although the definition of $MaxVote$ depends upon the ordering of votes, $B1(\mathcal{B})$ implies that $MaxVote(b, Q, \mathcal{B})_{dec}$ is independent of how votes with equal ballot numbers were ordered.

To show that these conditions imply consistency, the Paxons first showed that $B1(\mathcal{B})$–$B3(\mathcal{B})$ imply that, if a ballot B in $\mathcal{B}$ is successful, then any later ballot in $\mathcal{B}$ is for the same decree as B.

Lemma *If $B1(\mathcal{B})$, $B2(\mathcal{B})$, and $B3(\mathcal{B})$ hold, then*

$$((B_{qrm} \subseteq B_{vot}) \wedge (B'_{bal} > B_{bal})) \Rightarrow (B'_{dec} = B_{dec})$$

for any B, B' in $\mathcal{B}$.

Proof of Lemma

For any ballot B in $\mathcal{B}$, let $\Psi(B, \mathcal{B})$ be the set of ballots in $\mathcal{B}$ later than B for a decree different from B's:

$$\Psi(B, \mathcal{B}) \triangleq \{B' \in \mathcal{B} : (B'_{bal} > B_{bal}) \wedge (B'_{dec} \neq B_{dec})\}$$

To prove the lemma, it suffices to show that if $B_{qrm} \subseteq B_{vot}$ then $\Psi(B, \mathcal{B})$ is empty. The Paxons gave a proof by contradiction. They assumed the existence of a B with $B_{qrm} \subseteq B_{vot}$ and $\Psi(B, \mathcal{B}) \neq \emptyset$, and obtained a contradiction as follows.[9]

1. Choose $C \in \Psi(B, \mathcal{B})$ such that $C_{bal} = \min\{B'_{bal} : B' \in \Psi(B, \mathcal{B})\}$.
   PROOF: C exists because $\Psi(B, \mathcal{B})$ is nonempty and finite.
2. $C_{bal} > B_{bal}$
   PROOF: By 1 and the definition of $\Psi(B, \mathcal{B})$.
3. $B_{vot} \cap C_{qrm} \neq \emptyset$
   PROOF: By $B2(\mathcal{B})$ and the hypothesis that $B_{qrm} \subseteq B_{vot}$.

[8] I use the Paxon mathematical symbol $\triangleq$, which meant *equals by definition*.

[9] Paxon mathematicians always provided careful, structured proofs of important theorems. They were not as sophisticated as modern mathematicians, who can omit many details and write paragraph-style proofs without ever making a mistake.

4. $MaxVote(C_{bal}, C_{qrm}, \mathcal{B})_{bal} \geq B_{bal}$
   PROOF: By 2, 3 and the definition of $MaxVote(C_{bal}, C_{qrm}, \mathcal{B})$.
5. $MaxVote(C_{bal}, C_{qrm}, \mathcal{B}) \in Votes(\mathcal{B})$
   PROOF: By 4 (which implies that $MaxVote(C_{bal}, C_{qrm}, \mathcal{B})$ is not a null vote) and the definition of $MaxVote(C_{bal}, C_{qrm}, \mathcal{B})$.
6. $MaxVote(C_{bal}, C_{qrm}, \mathcal{B})_{dec} = C_{dec}$.
   PROOF: By 5 and $B3(\mathcal{B})$.
7. $MaxVote(C_{bal}, C_{qrm}, \mathcal{B})_{dec} \neq B_{dec}$
   PROOF: By 6, 1, and the definition of $\Psi(B, \mathcal{B})$.
8. $MaxVote(C_{bal}, C_{qrm}, \mathcal{B})_{bal} > B_{bal}$
   PROOF: By 4, since 7 and $B1(\mathcal{B})$ imply that $MaxVote(C_{bal}, C_{qrm}, \mathcal{B})_{bal} \neq B_{bal}$.
9. $MaxVote(C_{bal}, C_{qrm}, \mathcal{B}) \in Votes(\Psi(B, \mathcal{B}))$
   PROOF: By 7, 8, and the definition of $\Psi(B, \mathcal{B})$.
10. $MaxVote(C_{bal}, C_{qrm}, \mathcal{B})_{bal} < C_{bal}$
    PROOF: By definition of $MaxVote(C_{bal}, C_{qrm}, \mathcal{B})$.
11. Contradiction
    PROOF: By 9, 10, and 1.

End Proof of Lemma

With this lemma, it was easy to show that, if $B1$–$B3$ hold, then any two successful ballots are for the same decree.

THEOREM 1. *If $B1(\mathcal{B})$, $B2(\mathcal{B})$, and $B3(\mathcal{B})$ hold, then*

$$((B_{qrm} \subseteq B_{vot}) \wedge (B'_{qrm} \subseteq B'_{vot})) \Rightarrow (B'_{dec} = B_{dec})$$

for any B, B' in $\mathcal{B}$.

Proof of Theorem
If $B'_{bal} = B_{bal}$, then $B1(\mathcal{B})$ implies $B' = B$. If $B'_{bal} \neq B_{bal}$, then the theorem follows immediately from the lemma.
End Proof of Theorem

The Paxons then proved a theorem asserting that if there are enough priests in the Chamber, then it is possible to conduct a successful ballot while preserving $B1$–$B3$. Although this does not guarantee progress, it at least shows that a balloting protocol based on $B1$–$B3$ will not deadlock.

THEOREM 2. *Let b be a ballot number and Q a set of priests such that $b > B_{bal}$ and $Q \cap B_{qrm} \neq \emptyset$ for all $B \in \mathcal{B}$. If $B1(\mathcal{B})$, $B2(\mathcal{B})$, and $B3(\mathcal{B})$ hold, then there is a ballot B' with $B'_{bal} = b$ and $B'_{qrm} = B'_{vot} = Q$ such that $B1(\mathcal{B} \cup \{B'\})$, $B2(\mathcal{B} \cup \{B'\})$, and $B3(\mathcal{B} \cup \{B'\})$ hold.*

Proof of Theorem
Condition $B1(\mathcal{B} \cup \{B'\})$ follows from $B1(\mathcal{B})$, the choice of B'_{bal}, and the assumption about b. Condition $B2(\mathcal{B} \cup \{B'\})$ follows from $B2(\mathcal{B})$, the choice of B'_{qrm}, and the assumption about Q. If $MaxVote(b, Q, \mathcal{B})_{bal} = -\infty$ then let B'_{dec} be any decree, else let it equal $MaxVote(b, Q, \mathcal{B})_{dec}$. Condition $B3(\mathcal{B} \cup \{B'\})$ then follows from $B3(\mathcal{B})$.
End Proof of Theorem

2.2 The Preliminary Protocol

The Paxons derived the *preliminary protocol* from the requirement that conditions $B1(\mathcal{B})$–$B3(\mathcal{B})$ remain true, where $\mathcal{B}$ was the set of all ballots that had been or were being conducted. The definition of the protocol specified how the set $\mathcal{B}$ changed, but the set was never explicitly calculated. The Paxons referred to $\mathcal{B}$ as a quantity observed only by the gods, since it might never be known to any mortal.

Each ballot was initiated by a priest, who chose its number, decree, and quorum. Each priest in the quorum then decided whether or not to vote in the ballot. The rules determining how the initiator chose a ballot's number, decree, and quorum, and how a priest decided whether or not to vote in a ballot were derived directly from the need to maintain $B1(\mathcal{B})$–$B3(\mathcal{B})$.

To maintain $B1$, each ballot had to receive a unique number. By remembering (with notes in his ledger) what ballots he had previously initiated, a priest could easily avoid initiating two different ballots with the same number. To keep different priests from initiating ballots with the same number, the set of possible ballot numbers was partitioned among the priests. While it is not known how this was done, an obvious method would have been to let a ballot number be a pair consisting of an integer and a priest, using a lexicographical ordering, where

$$(13,\ \mathit{\Gamma\rho\alpha\breve{\iota}}) < (13,\ \mathit{\Lambda\iota\nu\sigma\epsilon\breve{\iota}}) < (15,\ \mathit{\Gamma\rho\alpha\breve{\iota}})$$

since $\mathit{\Gamma}$ came before $\mathit{\Lambda}$ in the Paxon alphabet. In any case, it is known that every priest had an unbounded set of ballot numbers reserved for his use.

To maintain $B2$, a ballot's quorum was chosen to contain a $\mu\alpha\delta\zeta\partial\omega\rho\iota\breve{\pi}\breve{\iota}\sigma\epsilon\tau$ of priests. Initially, $\mu\alpha\delta\zeta\partial\omega\rho\iota\breve{\pi}\breve{\iota}\sigma\epsilon\tau$ just meant a simple majority. Later, it was observed that fat priests were less mobile and spent more time in the Chamber than thin ones, so a $\mu\alpha\delta\zeta\partial\omega\rho\iota\breve{\pi}\breve{\iota}\sigma\epsilon\tau$ was taken to mean any set of priests whose total weight was more than half the total weight of all priests, rather than a simple majority of the priests. When a group of thin priests complained that this was unfair, actual weights were replaced with symbolic weights based on a priest's attendance record. The primary requirement for a $\mu\alpha\delta\zeta\partial\omega\rho\iota\breve{\pi}\breve{\iota}\sigma\epsilon\tau$ was that any two sets containing a $\mu\alpha\delta\zeta\partial\omega\rho\iota\breve{\pi}\breve{\iota}\sigma\epsilon\tau$ of priests had at least one priest in common. To maintain $B2$, the priest initiating a ballot B chose B_{qrm} to be a majority set.

Condition $B3$ requires that if $\mathit{MaxVote}(b, Q, \mathcal{B})_{dec}$ is not equal to BLANK, then a ballot with number b and quorum Q must have decree $\mathit{MaxVote}(b, Q, \mathcal{B})_{dec}$. If $\mathit{MaxVote}(b, Q, \mathcal{B})_{dec}$ equals BLANK, then the ballot can have any decree. To maintain $B3(\mathcal{B})$, before initiating a new ballot with ballot number b and quorum Q, a priest p had to find $\mathit{MaxVote}(b, Q, \mathcal{B})_{dec}$. To do this, p had to find $\mathit{MaxVote}(b, q, \mathcal{B})$ for each priest q in Q.

Recall that $\mathit{MaxVote}(b, q, \mathcal{B})$ is the vote with the largest ballot number less than b among all the votes cast by q, or null_q if q did not vote in any ballot numbered less than b. Priest p obtains $\mathit{MaxVote}(b, q, \mathcal{B})$ from q by an exchange of messages. Therefore, the first two steps in the protocol for conducting a single ballot initiated by p are:[10]

[10]Priests p and q could be the same. For simplicity, the protocol is described with p sending messages to himself in this case. In reality, a priest could talk to himself without the use of messengers.

(1) Priest p chooses a new ballot number b and sends a $NextBallot(b)$ message to some set of priests.
(2) A priest q responds to the receipt of a $NextBallot(b)$ message by sending a $LastVote(b, v)$ message to p, where v is the vote with the largest ballot number less than b that q has cast, or his null vote $null_q$ if q did not vote in any ballot numbered less than b.

Priest q must use notes in the back of his ledger to remember what votes he had previously cast.

When q sends the $LastVote(b, v)$ message, v equals $MaxVote(b, q, \mathcal{B})$. But the set $\mathcal{B}$ of ballots changes as new ballots are initiated and votes are cast. Since priest p is going to use v as the value of $MaxVote(b, q, \mathcal{B})$ when choosing a decree, to keep $B3(\mathcal{B})$ true it is necessary that $MaxVote(b, q, \mathcal{B})$ not change after q has sent the $LastVote(b, v)$ message. To keep $MaxVote(b, q, \mathcal{B})$ from changing, q must cast no new votes with ballot numbers between v_{bal} and b. By sending the $LastVote(b, v)$ message, q is promising not to cast any such vote. (To keep this promise, q must record the necessary information in his ledger.)

The next two steps in the balloting protocol (begun in step 1 by priest p) are:

(3) After receiving a $LastVote(b, v)$ message from every priest in some majority set Q, priest p initiates a new ballot with number b, quorum Q, and decree d, where d is chosen to satisfy $B3$. He then records the ballot in the back of his ledger and sends a $BeginBallot(b, d)$ message to every priest in Q.
(4) Upon receipt of the $BeginBallot(b, d)$ message, priest q decides whether or not to cast his vote in ballot number b. (He may not cast the vote if doing so would violate a promise implied by a $LastVote(b', v')$ message he has sent for some other ballot.) If q decides to vote for ballot number b, then he sends a $Voted(b, q)$ message to p and records the vote in the back of his ledger.

The execution of step 3 is considered to add a ballot B to $\mathcal{B}$, where $B_{bal} = b$, $B_{qrm} = Q$, $B_{vot} = \emptyset$ (no one has yet voted in this ballot), and $B_{dec} = d$. In step 4, if priest q decides to vote in the ballot, then executing that step is considered to change the set $\mathcal{B}$ of ballots by adding q to the set B_{vot} of voters in the ballot $B \in \mathcal{B}$.

A priest has the option not to vote in step 4, even if casting a vote would not violate any previous promise. In fact, all the steps in this protocol are optional. For example, a priest q can ignore a $NextBallot(b)$ message instead of executing step 2. Failure to take an action can prevent progress, but it cannot cause any inconsistency because it cannot make $B1(\mathcal{B})$–$B3(\mathcal{B})$ false. Since the only effect not receiving a message can have is to prevent an action from happening, message loss also cannot cause inconsistency. Thus, the protocol guarantees consistency even if priests leave the chamber or messages are lost.

Receiving multiple copies of a message can cause an action to be repeated. Except in step 3, performing the action a second time has no effect. For example, sending several $Voted(b, q)$ messages in step 4 has the same effect as sending just one. The repetition of step 3 is prevented by using the entry made in the back of the ledger when it is executed. Thus, the consistency condition is maintained even if a messenger delivers the same message several times.

Steps 1–4 describe the complete protocol for initiating a ballot and voting on it.

All that remains is to determine the results of the balloting and announce when a decree has been selected. Recall that a ballot is successful iff every priest in the quorum has voted. The decree of a successful ballot is the one chosen by the Synod. The rest of the protocol is:

(5) If p has received a $Voted(b, q)$ message from every priest q in Q (the quorum for ballot number b), then he writes d (the decree of that ballot) in his ledger and sends a $Success(d)$ message to every priest.

(6) Upon receiving a $Success(d)$ message, a priest enters decree d in his ledger.

Steps 1–6 describe how an individual ballot is conducted. The preliminary protocol allows any priest to initiate a new ballot at any time. Each step maintains $B1(\mathcal{B})$–$B3(\mathcal{B})$, so the entire protocol also maintains these conditions. Since a priest enters a decree in his ledger only if it is the decree of a successful ballot, Theorem 1 implies that the priests' ledgers are consistent. The protocol does not address the question of progress.

In step 3, if the decree d is determined by condition $B3$, then it is possible that this decree is already written in the ledger of some priest. That priest need not be in the quorum Q; he could have left the Chamber. Thus, consistency would not be guaranteed if step 3 allowed any greater freedom in choosing d.

2.3 The Basic Protocol

In the preliminary protocol, a priest must record (i) the number of every ballot he has initiated, (ii) every vote he has cast, and (iii) every *LastVote* message he has sent. Keeping track of all this information would have been difficult for the busy priests. The Paxons therefore restricted the preliminary protocol to obtain the more practical *basic protocol* in which each priest p had to maintain only the following information in the back of his ledger:

$lastTried[p]$ The number of the last ballot that p tried to initiate, or $-\infty$ if there was none.

$prevVote[p]$ The vote cast by p in the highest-numbered ballot in which he voted, or $-\infty$ if he never voted.

$nextBal[p]$ The largest value of b for which p has sent a $LastVote(b, v)$ message, or $-\infty$ if he has never sent such a message.

Steps 1–6 of the preliminary protocol describe how a single ballot is conducted by its initiator, priest p. The preliminary protocol allows p to conduct any number of ballots concurrently. In the basic protocol, he conducts only one ballot at a time—ballot number $lastTried[p]$. After p initiates this ballot, he ignores messages that pertain to any other ballot that he had previously initiated. Priest p keeps all information about the progress of ballot number $lastTried[p]$ on a slip of paper. If he loses that slip of paper, then he stops conducting the ballot.

In the preliminary protocol, each $LastVote(b, v)$ message sent by a priest q represents a promise not to vote in any ballot numbered between v_{bal} and b. In the basic protocol, it represents the stronger promise not to cast a new vote in any ballot numbered less than b. This stronger promise might prevent him from casting a vote in step 4 of the basic protocol that he would have been allowed to cast in the preliminary protocol. However, since the preliminary protocol always gives q

the option of not casting his vote, the basic protocol does not require him to do anything not allowed by the preliminary protocol.

Steps 1–6 of the preliminary protocol become the following six steps for conducting a ballot in the basic protocol. (All information used by p to conduct the ballot, other than $lastTried[p]$, $prevVote[p]$, and $nextBal[p]$, is kept on a slip of paper.)

(1) Priest p chooses a new ballot number b greater than $lastTried[p]$, sets $lastTried[p]$ to b, and sends a $NextBallot(b)$ message to some set of priests.

(2) Upon receipt of a $NextBallot(b)$ message from p with $b > nextBal[q]$, priest q sets $nextBal[q]$ to b and sends a $LastVote(b, v)$ message to p, where v equals $prevVote[q]$. (A $NextBallot(b)$ message is ignored if $b \leq nextBal[q]$.)

(3) After receiving a $LastVote(b, v)$ message from every priest in some majority set Q, where $b = lastTried[p]$, priest p initiates a new ballot with number b, quorum Q, and decree d, where d is chosen to satisfy $B3$. He then sends a $BeginBallot(b, d)$ message to every priest in Q.

(4) Upon receipt of a $BeginBallot(b, d)$ message with $b = nextBal[q]$, priest q casts his vote in ballot number b, sets $prevVote[q]$ to this vote, and sends a $Voted(b, q)$ message to p. (A $BeginBallot(b, d)$ message is ignored if $b \neq nextBal[q]$.)

(5) If p has received a $Voted(b, q)$ message from every priest q in Q (the quorum for ballot number b), where $b = lastTried[p]$, then he writes d (the decree of that ballot) in his ledger and sends a $Success(d)$ message to every priest.

(6) Upon receiving a $Success(d)$ message, a priest enters decree d in his ledger.

The basic protocol is a restricted version of the preliminary protocol, meaning that every action allowed by the basic protocol is also allowed by the preliminary protocol. Since the preliminary protocol satisfies the consistency condition, the basic protocol also satisfies that condition. Like the preliminary protocol, the basic protocol does not require that any action ever be taken, so it does not addresses the question of progress.

The derivation of the basic protocol from $B1$–$B3$ made it obvious that the consistency condition was satisfied. However, some similarly "obvious" ancient wisdom had turned out to be false, and skeptical citizens demanded a more rigorous proof. Their Paxon mathematicians' proof that the protocol satisfies the consistency condition is reproduced in the appendix.

2.4 The Complete Synod Protocol

The basic protocol maintains consistency, but it cannot ensure any progress because it states only what a priest may do; it does not require him to do anything. The complete protocol consists of the same six steps for conducting a ballot as the basic protocol. To help achieve progress, it includes the obvious additional requirement that priests perform steps 2–6 of the protocol as soon as possible. However, to meet the progress condition, it is necessary that some priest be required to perform step 1, which initiates a ballot. The key to the complete protocol lay in determining when a priest should initiate a ballot.

Never initiating a ballot will certainly prevent progress. However, initiating too may ballots can also prevent progress. If b is larger than any other ballot number, then the receipt of a $NextBallot(b)$ message by priest q in step 2 may elicit a promise that prevents him from voting in step 4 for any previously initiated ballot. Thus, the initiation of a new ballot can prevent any previously initiated ballot from succeeding. If new ballots are continually initiated with increasing ballot numbers before the previous ballots have a chance to succeed, then no progress might be made.

Achieving the progress condition requires that new ballots be initiated until one succeeds, but that they not be initiated too frequently. To develop the complete protocol, the Paxons first had to know how long it took messengers to deliver messages and priests to respond. They determined that a messenger who did not leave the Chamber would always deliver a message within 4 minutes, and a priest who remained in the Chamber would always perform an action within 7 minutes of the event that caused the action.[11] Thus, if p and q were in the Chamber when some event caused p to send a message to q, and q responded with a reply to p, then p would receive that reply within 22 minutes if neither messenger left the Chamber. (Priest p would send the message within 7 minutes of the event, q would receive the message within 4 more minutes, he would respond within 7 minutes, and the reply would reach p within 4 more minutes.)

Suppose that only a single priest p was initiating ballots, and that he did so by sending a message to every priest in step 1 of the protocol. If p initiated a ballot when a majority set of priests was in the chamber, then he could expect to execute step 3 within 22 minutes of initiating the ballot, and to execute step 5 within another 22 minutes. If he was unable to execute the steps by those times, then either some priest or messenger left the Chamber after p initiated the ballot, or a larger-numbered ballot had previously been initiated by another priest (before p became the only priest to initiate ballots). To handle the latter possibility, p had to learn about any ballot numbers greater than $lastTried[p]$ used by other priests. This could be done by extending the protocol to require that if a priest q received a $NextBallot(b)$ or a $BeginBallot(b, d)$ message from p with $b < nextBal[q]$, then he sent p a message containing $nextBal[q]$. Priest p would then initiate a new ballot with a larger ballot number.

Still assuming that p was the only priest initiating ballots, suppose that he were required to initiate a new ballot iff (i) he had not executed step 3 or step 5 within the previous 22 minutes, or (ii) he learned that another priest had initiated a higher-numbered ballot. If the Chamber doors were locked with p and a majority set of priests inside, then a decree would be passed and recorded in the ledgers of all priests in the Chamber within 99 minutes. (It could take 22 minutes for p to start the next ballot, 22 more minutes to learn that another priest had initiated a larger-numbered ballot, then 55 minutes to complete steps 1–6 for a successful ballot.) Thus, the progress condition would be met if only a single priest, who did not leave the chamber, were initiating ballots.

The complete protocol therefore included a procedure for choosing a single priest, called the *president*, to initiate ballots. In most forms of government, choosing a

[11] I am assuming a value of 30 seconds for the $\delta\zeta\partial\iota\phi\breve{\iota}$, the Paxon unit of time. This value is within the range determined from studies of hourglass shards. The reaction time of priests was so long because they had to respond to every message within 7 minutes (14 $\delta\zeta\partial\iota\phi\breve{\iota}$), even if a number of messages arrived simultaneously.

president can be a difficult problem. However, the difficultly arises only because most governments require that there be exactly one president at any time. In the United States, for example, chaos would result if some people thought Bush had been elected president while others thought that Dukakis had, since one of them might decide to sign a bill into law while the other decided to veto it. However, in the Paxon Synod, having multiple presidents could only impede progress; it could not cause inconsistency. For the complete protocol to satisfy the progress condition, the method for choosing the president needed only to satisfy the following *presidential selection requirement*:

> If no one entered or left the Chamber, then after T minutes exactly one priest in the Chamber would consider himself to be the president.

If the presidential selection requirement were met, then the complete protocol would have the property that if a majority set of priests were in the chamber and no one entered or left the Chamber for $T + 99$ minutes, then at the end of that period every priest in the Chamber would have a decree written in his ledger.

The Paxons chose as president the priest whose name was last in alphabetical order among the names of all priests in the Chamber, though we don't know exactly how this was done. The presidential selection requirement would have been satisfied if a priest in the Chamber sent a message containing his name to every other priest at least once every $T - 11$ minutes, and a priest considered himself to be president iff he received no message from a "higher-named" priest for T minutes.

The complete Synod protocol was obtained from the basic protocol by requiring priests to perform steps 2–6 promptly, adding a method for choosing a president who initiated ballots, and requiring the president to initiate ballots at the appropriate times. Many details of the protocol are not known. I have described simple methods for selecting a president and for deciding when the president should initiate a new ballot, but they are undoubtedly not the ones used in Paxos. The rules I have given require the president to keep initiating ballots even after a decree has been chosen, thereby ensuring that priests who have just entered the Chamber learn about the chosen decree. There were obviously better ways to make sure priests learned about the decree after it had been chosen. Also, in the course of selecting a president, each priest probably sent his value of $lastTried[p]$ to the other priests, allowing the president to choose a large enough ballot number on his first try.

The Paxons realized that any protocol to achieve the progress condition must involve measuring the passage of time.[12] The protocols given above for selecting a president and initiating ballots are easily formulated as precise algorithms that set timers and perform actions when time-outs occur—assuming perfectly accurate timers. A closer analysis reveals that such protocols can be made to work with timers having a known bound on their accuracy. The skilled glass blowers of Paxos had no difficulty constructing suitable hourglass timers.

Given the sophistication of Paxon mathematicians, it is widely believed that they must have found an optimal algorithm to satisfy the presidential selection requirement. We can only hope that this algorithm will be discovered in future

[12] However, many centuries were to pass before a rigorous proof of this result was given.[Fischer et al. 1985]

excavations on Paxos.

3 The Multi-Decree Parliament

When Parliament was established, a protocol to satisfy its consistency and progress requirements was derived from the Synod protocol. The derivation and properties of the original parliamentary protocol are described in Sections 3.1 and 3.2. Section 3.3 discusses the further evolution of the protocol.

3.1 The Protocol

Instead of passing just one decree, the Paxon Parliament had to pass a series of numbered decrees. As in the Synod protocol, a president was elected. Anyone who wanted a decree passed would inform the president, who would assign a number to the decree and attempt to pass it. Logically, the parliamentary protocol used a separate instance of the complete Synod protocol for each decree number. However, a single president was selected for all these instances, and he performed the first two steps of the protocol just once.

The key to deriving the parliamentary protocol is the observation that, in the Synod protocol, the president does not choose the decree or the quorum until step 3. A newly elected president p can send to some set of legislators a single message that serves as the $NextBallot(b)$ message for all instances of the Synod protocol. (There are an infinite number of instances—one for each decree number.) A legislator q can reply with a single message that serves as the $LastVote$ messages for step 2 of all instances of the Synod protocol. This message contains only a finite amount of information, since q can have voted in only a finite number of instances.

When the new president has received a reply from every member of a majority set, he is ready to perform step 3 for every instance of the Synod protocol. For some finite number of instances (decree numbers), the choice of decree in step 3 will be determined by $B3$. The president immediately performs step 3 for each of those instances to try passing these decrees. Then, whenever he receives a request to pass a decree, he chooses the lowest-numbered decree that he is still free to choose, and he performs step 3 for that decree number (instance of the Synod protocol) to try to pass the decree.

The following modifications to this simple protocol lead to the actual Paxon Parliament's protocol.

—There is no reason to go through the Synod protocol for a decree number whose outcome is already known. Therefore, if a newly elected president p has all decrees with numbers less than or equal to n written in his ledger, then he sends a $NextBallot(b, n)$ message that serves as a $NextBallot(b)$ message in all instances of the Synod protocol for decree numbers larger than n. In his response to this message, legislator q informs p of all decrees numbered greater than n that already appear in q's ledger (in addition to sending the usual $LastVote$ information for decrees not in his ledger), and he asks p to send him any decrees numbered n or less that are not in his ledger.

—Suppose decrees 125 and 126 are introduced late Friday afternoon, decree 126 is passed and is written in one or two ledgers, but before anything else happens, the legislators all go home for the weekend. Suppose also that the following Monday,

$\Delta\phi\omega\rho\kappa$ is elected the new president and learns about decree 126, but she has no knowledge of decree 125 because the previous president and all legislators who had voted for it are still out of the Chamber. She will hold a ballot that passes decree 126, which leaves a gap in the ledgers. Assigning number 125 to a new decree would cause it to appear earlier in the ledger than decree 126, which had been passed the previous week. Passing decrees out of order in this way might cause confusion—for example, if the citizen who proposed the new decree did so because he knew decree 126 had already passed. Instead, $\Delta\phi\omega\rho\kappa$ would attempt to pass

125: *The ides of February is national olive day*

a traditional decree that made absolutely no difference to anyone in Paxos. In general, a new president would fill any gaps in his ledger by passing the "olive-day" decree.

The consistency and progress properties of the parliamentary protocol follow immediately from the corresponding properties of the Synod protocol from which it was derived. To our knowledge, the Paxons never bothered writing a precise description of the parliamentary protocol because it was so easily derived from the Synod protocol.

3.2 Properties of the Protocol

3.2.1 *The Ordering of Decrees* Balloting could take place concurrently for many different decree numbers, with ballots initiated by different legislators—each thinking he was president when he initiated the ballot. We cannot say precisely in what order decrees would be passed, especially without knowing how a president was selected. However, there is one important property about the ordering of decrees that can be deduced.

A decree was said to to be *proposed* when it was chosen by the president in step 3 of the corresponding instance of the Synod protocol. The decree was said to be *passed* when it was written for the first time in a ledger. Before a president could propose any new decrees, he had to learn from all the members of a majority set what decrees they had voted for. Any decree that had already been passed must have been voted for by at least one legislator in the majority set. Therefore, the president must have learned about all previously passed decrees before initiating any new decree. The president would not fill a gap in the ledgers with an important decree—that is, with any decree other than the "olive-day" decree. He would also not propose decrees out of order. Therefore, the protocol satisfied the following *decree-ordering property.*

> If decrees A and B are important and decree A was passed before decree B was proposed, then A has a lower decree number than B.

3.2.2 *Behind Closed Doors* Although we don't know the details involved in choosing a new president, we do know exactly how Parliament functioned when the president had been chosen and no one was entering or leaving the Chamber. Upon receiving a request to pass a decree—either directly from a citizen or relayed from another legislator—the president assigned the decree a number and passed it with the following exchange of messages. (The numbers refer to the corresponding steps in the Synod protocol.)

(3) The president sent a *BeginBallot* message to each legislator in a quorum.
(4) Each legislator in the quorum sent a *Voted* message to the president.
(5) The president sent a *Success* message to every legislator.

This is a total of three message delays and about $3N$ messages, assuming a parliament of N legislators and a quorum of about $N/2$. Moreover, if Parliament was busy, the president would combine the *BeginBallot* message for one decree with the *Success* message for a previous one, for a total of only $2N$ messages per decree.

3.3 Further Developments

Governing the island turned out to be a more complex task than the Paxons realized. A number of problems arose whose solutions required changes to the protocol. The most important of these changes are described below.

3.3.1 *Picking a President* The president of parliament was originally chosen by the method that had been used in the Synod, which was based purely on the alphabetical ordering of names. Thus, when legislator $\Omega\kappa\iota$ returned from a six-month vacation, he was immediately made president—even though he had no idea what had happened in his absence. Parliamentary activity came to a halt while $\Omega\kappa\iota$, who was a slow writer, laboriously copied six months worth of decrees to bring his ledger up to date.

This incident led to a debate about the best way to choose a president. Some Paxons urged that once a legislator became president, he should remain president until he left the Chamber. An influential group of citizens wanted the richest legislator in the Chamber to be president, since he could afford to hire more scribes and other servants to help him with the presidential duties. They argued that once a rich legislator had brought his ledger up to date, there was no reason for him not to assume the presidency. Others, however, argued that the most upstanding citizen should be made president, regardless of wealth. Upstanding probably meant less likely to be dishonest, although no Paxon would publicly admit the possibility of official malfeasance. Unfortunately, the outcome of this debate is not known; no record exists of the presidential selection protocol that was ultimately used.

3.3.2 *Long Ledgers* As the years progressed and Parliament passed more and more decrees, Paxons had to pore over an ever longer list of decrees to find the current olive tax or what color goat could be sold. A legislator who returned to the Chamber after an extended voyage had to do quite a bit of copying to bring his ledger up to date. Eventually, the legislators were forced to convert their ledgers from lists of decrees into law books that contained only the current state of the law and the number of the last decree whose passage was reflected in that state.

To learn the current olive tax, one looked in the law book under “taxes”; to learn what color goat could be sold, one looked under “mercantile law”. If a legislator’s ledger contained the law through decree 1298 and he learned that decree 1299 set the olive tax to 6 drachmas per ton, he just changed the entry for the olive-tax law and noted that his ledger was complete through decree 1299. If he then learned about decree 1302, he would write it down in the back of the ledger and wait until he learned about decrees 1300 and 1301 before incorporating decree 1302 into the law book.

To enable a legislator who had been gone for a short time to catch up without copying the entire law book, legislators kept a list of the past week's decrees in the back of the book. They could have kept this list on a slip of paper, but it was convenient for a legislator to enter decrees in the back of the ledger as they were passed and update the law book only two or three times a week.

3.3.3 *Bureaucrats* As Paxos prospered, legislators became very busy. Parliament could no longer handle all details of government, so a bureaucracy was established. Instead of passing a decree to declare whether each lot of cheese was fit for sale, Parliament passed a decree appointing a cheese inspector to make those decisions.

It soon became evident that selecting bureaucrats was not as simple as it first seemed. Parliament passed a decree making $\Delta\check{\iota}\kappa\sigma\tau\rho\alpha$ the first cheese inspector. After some months, merchants complained that $\Delta\check{\iota}\kappa\sigma\tau\rho\alpha$ was too strict and was rejecting perfectly good cheese. Parliament then replaced him by passing the decree

1375: *$\Gamma\omega\upsilon\delta\alpha$ is the new cheese inspector*

But $\Delta\check{\iota}\kappa\sigma\tau\rho\alpha$ did not pay close attention to what Parliament did, so he did not learn of this decree right away. There was a period of confusion in the cheese market when both $\Delta\check{\iota}\kappa\sigma\tau\rho\alpha$ and $\Gamma\omega\upsilon\delta\alpha$ were inspecting cheese and making conflicting decisions.

To prevent such confusion, the Paxons had to guarantee that a position could be held by at most one bureaucrat at any time. To do this, a president included as part of each decree the time and date when it was proposed. A decree making $\Delta\check{\iota}\kappa\sigma\tau\rho\alpha$ the cheese inspector might read

2716: *8:30 15 Jan 72—$\Delta\check{\iota}\kappa\sigma\tau\rho\alpha$ is cheese inspector for 3 months*

This declares his term to begin either at 8:30 on 15 January or when the previous inspector's term ended—whichever was later. His term would end at 8:30 on 15 March, unless he explicitly resigned by asking the president to pass a decree like

2834: *9:15 3 Mar 72—$\Delta\check{\iota}\kappa\sigma\tau\rho\alpha$ resigns as cheese inspector*

A bureaucrat was appointed for a short term, so he could be replaced quickly—for example, if he left the island. Parliament would pass a decree to extend the bureaucrat's term if he was doing a satisfactory job.

A bureaucrat needed to tell time to determine if he currently held a post. Mechanical clocks were unknown on Paxos, but Paxons could tell time accurately to within 15 minutes by the position of the sun or the stars.[13] If $\Delta\check{\iota}\kappa\sigma\tau\rho\alpha$'s term began at 8:30, he would not start inspecting cheese until his celestial observations indicated that it was 8:45.

It is easy to make this method of appointing bureaucrats work if higher-numbered decrees always have later proposal times. But what if Parliament passed the decrees

2854: *9:45 9 Apr 78—$\Phi\rho\alpha\nu\sigma\epsilon\zeta$ is wine taster for 2 months*

2855: *9:20 9 Apr 78—$\Pi\upsilon\nu\epsilon\lambda\check{\iota}$ is wine taster for 1 month*

[13]Cloudy days are rare in Paxos's balmy climate.

that were proposed between 9:30 and 9:35 by different legislators who both thought they were president? Such out-of-order proposal times are easily prevented because the parliamentary protocol satisfies the following property.

> If two decrees are passed by different presidents, then one of the presidents proposed his decree after learning that the other decree had been proposed.

To see that this property is satisfied, suppose that ballot number b was successful for decree D, ballot number b' was successful for decree D', and $b < b'$. Let q be a legislator who voted in both ballots. The balloting for D' began with a $NextBallot(b', n)$ message. If the sender of that message did not already know about D, then n is less than the decree number of D, and q's reply to the $NextBallot$ message must state that he voted for D.

3.3.4 *Learning the Law* In addition to requesting the passage of decrees, ordinary citizens needed to inquire about the current law of the land. The Paxons at first thought that a citizen could simply examine the ledger of any legislator, but the following incident demonstrated that a more sophisticated approach was needed. For centuries, it had been legal to sell only white goats. A farmer named $\Delta\omega\lambda\epsilon\phi$ got Parliament to pass the decree

77: *The sale of black goats is permitted*

$\Delta\omega\lambda\epsilon\phi$ then instructed his goatherd to sell some black goats to a merchant named $\Sigma\kappa\epsilon\epsilon\nu$. As a law-abiding citizen, $\Sigma\kappa\epsilon\epsilon\nu$ asked legislator $\Sigma\pi\omega\kappa\mu\epsilon\breve{\iota}\rho$ if such a sale would be legal. But $\Sigma\pi\omega\kappa\mu\epsilon\breve{\iota}\rho$ had been out of the Chamber and had no entry in his ledger past decree 76. He advised $\Sigma\kappa\epsilon\epsilon\nu$ that the sale would be illegal under the current law, so $\Sigma\kappa\epsilon\epsilon\nu$ refused to buy the goats.

This incident led to the formulation of the following *monotonicity condition* on inquiries about the law.

> If one inquiry precedes a second inquiry, then the second inquiry cannot reveal an earlier state of the law than the first.

If a citizen learns that a particular decree has been passed, then the process of acquiring that knowledge is considered to be an implicit inquiry to which this condition applies. As we will see, the interpretation of the monotonicity condition changed over the years.

Initially, the monotonicity condition was achieved by passing a decree for each inquiry. If $\Sigma\partial\nu\breve{\iota}\delta\epsilon\rho$ wanted to know the current tax on olives, he would get Parliament to pass a decree such as

87: *Citizen* $\Sigma\partial\nu\breve{\iota}\delta\epsilon\rho$ *is reading the law*

He would then read any ledger complete at least through decree 86 to learn the olive tax as of that decree. If citizen $\Gamma\rho\epsilon\epsilon\varsigma$ then inquired about the olive tax, the decree for his inquiry was proposed after decree 87 was passed, so the decree-ordering property (Section 3.2.1) implies that it received a decree number greater than 87. Therefore, $\Gamma\rho\epsilon\epsilon\varsigma$ could not obtain an earlier value of the olive tax than $\Sigma\partial\nu\breve{\iota}\delta\epsilon\rho$.

This method of reading the law satisfied the monotonicity condition when *precedes* was interpreted to mean that inquiry A precedes inquiry B iff A finished at an earlier time than B began.

Passing a decree for every inquiry soon proved too cumbersome. The Paxons realized that a simpler method was possible if they weakened the monotonicity condition by changing the interpretation of *precedes*. They decided that for one event to precede another, the first event not only had to happen at an earlier time, but it had to be able to causally affect the second event. The weaker monotonicity condition prevents the problem first encountered by farmer $\Delta\omega\lambda\epsilon\phi$ and merchant $\Sigma\kappa\epsilon\epsilon\nu$ because there is a causal chain of events between the end of the implicit inquiry by $\Delta\omega\lambda\epsilon\phi$ and the beginning of the inquiry by $\Sigma\kappa\epsilon\epsilon\nu$.

The weaker monotonicity condition was met by using decree numbers in all business transactions and inquiries. For example, farmer $\Delta\omega\lambda\epsilon\phi$, whose flock included many nonwhite goats, got Parliament to pass the decree

277: *The sale of brown goats is permitted*

When selling his brown goats to $\Sigma\kappa\epsilon\epsilon\nu$, he informed the merchant that the sale was legal as of decree number 277. $\Sigma\kappa\epsilon\epsilon\nu$ then asked legislator $\Sigma\pi\omega\kappa\mu\epsilon\breve{\iota}\rho$ if the sale were legal under the law through at least decree 277. If $\Sigma\pi\omega\kappa\mu\epsilon\breve{\iota}\rho$'s ledger was not complete through decree 277, he would either wait until it was or else tell $\Sigma\kappa\epsilon\epsilon\nu$ to ask someone else. If $\Sigma\pi\omega\kappa\mu\epsilon\breve{\iota}\rho$'s ledger went through decree 298, then he would tell $\Sigma\kappa\epsilon\epsilon\nu$ that the sale was legal as of decree number 298. Merchant $\Sigma\kappa\epsilon\epsilon\nu$ would remember the number 298 for use in his next business transaction or inquiry about the law.

The Paxons had satisfied the monotonicity condition, but ordinary citizens disliked having to remember decree numbers. Again, the Paxons solved the problem by re-interpreting the monotonicity condition—this time, by changing the meaning of *state of the law*. They divided the law into separate areas, and a legislator was chosen as specialist for each area. The current state of each area of the law was determined by that specialist's ledger. For example, suppose decree 1517 changed the tariff law and decree 1518 changed the tax law. The tax law would change first if the tax-law specialist learned of both decrees before the tariff-law specialist learned of either, yielding a state of the law that could not be obtained by enacting the decrees in numerical order.

To avoid conflicting definitions of the current state, the Paxons required that there be at most one specialist at a time for any area. This requirement was satisfied by using the same method to choose specialists that was used to choose bureaucrats (see Section 3.3.3). If each inquiry involved only a single area of the law, monotonicity was then achieved by directing the inquiry to that area's specialist, who answered it from his ledger. Since learning that a law had passed constituted the result of an implicit inquiry, the Paxons required that a decree change at most one area of the law, and that notification of the decree's passage could come only from the area's specialist.

Inquiries involving multiple areas were not hard to handle. When merchant $\Lambda\iota\sigma\kappa\omega\phi$ asked if the tariff on an imported golden fleece was higher than the sales tax on one purchased locally, the tax-law and tariff-law specialists had to cooperate to provide an answer. For example, the tax specialist could answer $\Lambda\iota\sigma\kappa\omega\phi$ by first asking the tariff specialist for the tariff on golden fleeces, so long as he made no changes to his ledger before receiving a reply.

This method proved satisfactory until it became necessary to make a sweeping change to several areas of the law at one time. The Paxons then realized that the necessary requirement for maintaining monotonicity was not that a decree affect only a single area, but that every area it affects have the same specialist. Parliament could change several areas of the law with a single decree by first appointing a single legislator to be the specialist for all those areas. Moreover, the same area could have multiple specialists, so long as that area of the law was not allowed to change. Just before income taxes were due, Parliament would appoint several tax-law specialists to handle the seasonal flood of inquiries about the tax law.

3.3.5 *Dishonest Legislators and Honest Mistakes* Despite official assertions to the contrary, there must have been a few dishonest legislators in the history of Paxos. When caught, they were probably exiled. By sending contradictory messages, a malicious legislator could cause different legislators' ledgers to be inconsistent. Inconsistency could also result from a lapse of memory by an honest legislator or messenger.

When inconsistencies were recognized, they could easily be corrected by passing decrees. For example, disagreement about the current olive tax could be eliminated by passing a new decree declaring the tax to have a certain value. The difficult problem lay in correcting inconsistent ledgers even if no one was aware of the inconsistency.

The existence of dishonesty or mistakes by legislators can be inferred from the redundant decrees that began appearing in ledgers several years after the founding of Parliament. For example, the decree

2605: *The olive tax is 9 drachmas per ton*

was passed even though decree 2155 had already set the olive tax to 9 drachmas per ton, and no intervening decree had changed it. Parliament apparently cycled through its laws every six months so that even if legislators' ledgers were initially inconsistent, all legislators would agree on the current law of the land within six months. It is believed that by the use of these redundant decrees, the Paxons made their Parliament self-stabilizing. (*Self-stabilizing* is a modern term due to Dijkstra [Dijkstra 1974].)

It is not clear precisely what self-stabilization meant in a Parliament with legislators coming and going at will. The Paxons would not have been satisfied with a definition that required all legislators to be in the Chamber at one time before consistency could be guaranteed. However, achieving consistency required that if one legislator had an entry in his ledger for a certain decree number and another did not, then the second legislator would eventually fill in that entry.

Unfortunately, we don't know exactly what sort of self-stabilization property the Paxon Parliament possessed or how it was achieved. Paxon mathematicians undoubtedly addressed the problem, but their work has not yet been found. I hope that future archaeological expeditions to Paxos will give high priority to the search for manuscripts on self-stabilization.

3.3.6 *Choosing New Legislators* At first, membership in Parliament was hereditary, passing from parent to child. When the elder statesman $\Pi\alpha\rho\nu\alpha\varsigma$ retired, he gave his ledger to his son, who carried on without interruption. It made no difference to other legislators which $\Pi\alpha\rho\nu\alpha\varsigma$ they communicated with.

As old families emigrated and new ones immigrated, this system had to change. The Paxons decided to add and remove members of Parliament by decree. This posed a circularity problem: membership in Parliament was determined by which decrees were passed, but passing a decree required knowing what constituted a majority set, which in turn depended upon who was a member of Parliament. The circularity was broken by letting the membership of Parliament used in passing decree n be specified by the law as of decree $n-3$. A president could not try to pass decree 3255 until he knew all decrees through decree 3252. In practice, after passing the decree

3252: $\Sigma\tau\rho\omega\nu\gamma$ *is now a legislator*

the president would immediately pass the "olive-day" decree as decrees 3253 and 3254.

Changing the composition of Parliament in this way was dangerous and had to be done with care. The consistency and progress conditions would always hold. However, the progress condition guaranteed progress only if a majority set was in the Chamber; it did not guarantee that a majority set would ever be there. In fact, the mechanism for choosing legislators led to the downfall of the Parliamentary system in Paxos. Because of a scribe's error, a decree that was supposed to honor sailors who had drowned in a shipwreck instead declared them to be the only members of Parliament. Its passage prevented any new decrees from being passed—including the decrees proposed to correct the mistake. Government in Paxos came to a halt. A general named $\Lambda\alpha\mu\pi\sigma\omega\nu$ took advantage of the confusion to stage a coup, establishing a military dictatorship that ended centuries of progressive government. Paxos grew weak under a series of corrupt dictators, and was unable to repel an invasion from the east that led to the destruction of its civilization.

4 Relevance to Computer Science

4.1 The State Machine Approach

Although Paxos's Parliament was destroyed many centuries ago, its protocol is still useful. For example, consider a simple distributed database system that might be used as a name server. A state of the database consists of an assignment of values to names. Copies of the database are maintained by multiple servers. A client program can issue, to any server, a request to read or change the value assigned to a name. There are two kinds of read request: a *slow read*, which returns the value currently assigned to a name, and a *fast read*, which is faster but might not reflect a recent change to the database.

There is an obvious correspondence between this database system and the Paxon Parliament:

Parliament		Distributed Database
legislator	$\leftrightarrow$	server
citizen	$\leftrightarrow$	client program
current law	$\leftrightarrow$	database state

command:	**read**(*name*, *client*)	**update**(*name*, *val*, *client*)
response:	(*client*, value of *name*)	(*client*, "ok")
new state:	Same as current state	Same as current state except value of *name* changed to *val*

Fig. 2. State machine for simple database.

A client's request to change a value is performed by passing a decree. A *slow read* involves passing a decree, as described in Section 3.3.4. A *fast read* is performed by reading the server's current version of the database. The Paxon Parliament protocol provides a distributed, fault-tolerant implementation of the database system,

This method of implementing a distributed database is an instance of the state machine approach, first proposed in [Lamport 1978]. In this approach, one first defines a *state machine*, which consists of a set of states, a set of commands, a set of responses, and a function that assigns a response/state pair (a pair consisting of a response and a state) to each command/state pair. Intuitively, a state machine executes a command by producing a response and changing its state; the command and the machine's current state determine its response and its new state. For the distributed database, a state-machine state is just a database state. The state-machine commands and the function specifying the response and new state are described in Figure 2.

In the state-machine approach, a system is implemented with a network of server processes. The servers transform client requests into state machine commands, execute the commands, and transform the state-machine responses into replies to clients. A general algorithm ensures that all servers obtain the same sequence of commands, thereby ensuring that they all produce the same sequence of responses and state changes—assuming they all start from the same initial state. In the database example, a client request to perform a *slow read* or to change a value is transformed into a state-machine **read** or **update** command. That command is executed, and the state-machine response is transformed into a reply to the client, which is sent to him by the server who received his request. Since all servers perform the same sequence of state-machine commands, they all maintain consistent versions of the database. However, at any time, some servers may have earlier versions than others because a state-machine command need not be executed at the same time by all servers. A server uses his current version of the state to reply to a *fast read* request, without executing a state-machine command.

The functionality of the system is expressed by the state machine, which is just a function from command/state pairs to response/state pairs. Problems of synchronization and fault-tolerance are handled by the general algorithm with which servers obtain the sequence of commands. When designing a new system, only the state machine is new. The servers obtain the state-machine commands by a standard distributed algorithm that has already been proved correct. Functions are much easier to design, and to get right, than distributed algorithms.

The first algorithm for implementing an arbitrary state machine appeared in [Lamport 1978]. Later, algorithms were devised to tolerate up to any fixed number f of arbitrary failures [Lamport 1984]. These algorithms guarantee that, if fewer than f processes fail, then state machine commands are executed within a fixed length of time. The algorithms are thus suitable for applications requiring real-time response.[14] But if more than f failures occur, then different servers may have inconsistent copies of the state machine. Moreover, the inability of two servers to communicate with each other is equivalent to the failure of one of them. For a system to have a low probability of losing consistency, it must use an algorithm with a large value of f, which in turn implies a large cost in redundant hardware, communication bandwidth, and response time.

The Paxon Parliament's protocol provides another way to implement an arbitrary state machine. The legislators' law book corresponds to the machine state, and passing a decree corresponds to executing a state-machine command. The resulting algorithm is less robust and less expensive than the earlier algorithms. It does not tolerate arbitrary, malicious failures, nor does it guarantee bounded-time response. However, consistency is maintained despite the (benign) failure of any number of processes and communication paths. The Paxon algorithm is suitable for systems with modest reliability requirements that do not justify the expense of an extremely fault-tolerant, real-time implementation.

If the state machine is executed with an algorithm that guarantees bounded-time response, then time can be made part of the state, and machine actions can be triggered by the passage of time. For example, consider a system for granting ownership of resources. The state can include the time at which a client was granted a resource, and the state machine can automatically execute a command to revoke ownership if the client has held the resource too long.

With the Paxon algorithm, time cannot be made part of the state in such a natural way. If failures occur, it can take arbitrarily long to execute a command (pass a decree), and one command can be executed before (appear earlier in the sequence of decrees than) another command that was issued earlier. However, a state machine can still use real time the same way the Paxon Parliament did. For example, the method described in Section 3.3.3 for deciding who was the current cheese inspector can be used to decide who is the current owner of a resource.

4.2 Commit Protocols

The Paxon Synod protocol is similar to standard three-phase commit protocols [Bernstein et al. 1987; Skeen 1982]. A Paxon ballot and a three-phase commit protocol both involve the exchange of five messages between a coordinator (the president) and the other quorum members (legislators). A commit protocol chooses one of two values—*commit* or *abort*—while the Synod protocol chooses an arbitrary decree. To convert a commit protocol to a Synod protocol, one sends the decree in the initial round of messages. A *commit* decision means that this decree was passed, and an *abort* decision means that the "olive-day" decree was passed.

The Synod protocol differs from a converted commit protocol because the decree is not sent until the second phase. This allows the corresponding parliamentary protocol to execute the first phase just once for all decrees, so the exchange of only three messages is needed to pass each individual decree.

The theorems on which the Synod protocol is based are similar to results obtained

[14] These algorithms were derived from the military protocols of another Mediterranean state.

by Dwork, Lynch, and Stockmeyer [Dwork et al. 1988]. However, their algorithms execute ballots sequentially in separate rounds, and they seem to be unrelated to the Synod protocol.

Much research has been done in the field since this article was written. The state-machine approach has been surveyed by Schneider [1990]. The recovery protocol by Keidar and Dolev [1996] and the totally-ordered broadcast algorithm of Fekete et al. [1997] are quite similar to the Paxon protocol described here. The author was also apparently unaware that the view management protocol by Oki and Liskov [1988] seems to be equivalent to the Paxon protocol.

Many of the refinements presented in this submission have also appeared in contemporary or subsequent articles. The method of delegation described in Section 3.3.3 is very similar to the leases mechanism of Gray and Cheriton [1989]. The technique of Section 3.3.4 in which the Paxons satisfy the monotonicity condition by using decree numbers is described by Ladin et al. [1992]. The technique of Section 3.3.6 for adding new legislators was also given by Schneider [1990].

K. M.

Appendix: Proof of Consistency of the Synodic Protocol

A1 The Basic Protocol

The Synod's basic protocol, described informally in Section 2.3, is stated here using modern algorithmic notation. We begin with the variables that a priest p must maintain. First come the variables that represent information kept in his ledger. (For convenience, the vote $prevVote[p]$ used in Section 2.3 is replaced by its components $prevBal[p]$ and $prevDec[p]$.)

- $outcome[p]$ The decree written in p's ledger, or BLANK if there is nothing written there yet.
- $lastTried[p]$ The number of the last ballot that p tried to begin, or $-\infty$ if there was none.
- $prevBal[p]$ The number of the last ballot in which p voted, or $-\infty$ if he never voted.
- $prevDec[p]$ The decree for which p last voted, or BLANK if p never voted.
- $nextBal[p]$ The number of the last ballot in which p agreed to participate, or $-\infty$ if he has never agreed to participate in a ballot.

Next come variables representing information that priest p could keep on a slip of paper:

- $status[p]$ One of the following values:
  - *idle* Not conducting or trying to begin a ballot
  - *trying* Trying to begin ballot number $lastTried[p]$
  - *polling* Now conducting ballot number $lastTried[p]$

  If p has lost his slip of paper, then $status[p]$ is assumed to equal *idle* and the values of the following four variables are irrelevant.
- $prevVotes[p]$ The set of votes received in *LastVote* messages for the current ballot (the one with ballot number $lastTried[p]$).

quorum[p]	If *status*[p] $=$ *polling*, then the set of priests forming the quorum of the current ballot; otherwise, meaningless.
voters[p]	If *status*[p] $=$ *polling*, then the set of quorum members from whom p has received *Voted* messages in the current ballot; otherwise, meaningless.
decree[p]	If *status*[p] $=$ *polling*, then the decree of the current ballot; otherwise, meaningless.

There is also the history variable $\mathcal{B}$, which is the set of ballots that have been started and their progress—namely, which priests have cast votes. (A history variable is one used in the development and proof of an algorithm, but not actually implemented.)

Next come the actions that priest p may take. These actions are assumed to be atomic, meaning that once an action is begun, it must be completed before priest p begins any other action. An action is described by an enabling condition and a list of effects. The enabling condition describes when the action can be performed; actions that receive a message are enabled whenever a messenger has arrived with the appropriate message. The list of effects describes how the action changes the algorithm's variables and what message, if any, it sends. (Each individual action sends at most one message.)

Recall that ballot numbers were partitioned among the priests. For any ballot number b, the Paxons defined $owner(b)$ to be the priest who was allowed to use that ballot number.

The actions in the basic protocol are allowed actions; the protocol does not require that a priest ever do anything. No attempt at efficiency has been made; the actions allow p to do silly things, such as sending another *BeginBallot* message to a priest from whom he has already received a *LastVote* message.

Try New Ballot

Always enabled.

– Set *lastTried*[p] to any ballot number b, greater than its previous value, such that $owner(b) = p$.
– Set *status*[p] to *trying*.
– Set *prevVotes*[p] to $\emptyset$.

Send *NextBallot* **Message**

Enabled whenever *status*[p]$=$*trying*.

– Send a $NextBallot(lastTried[p])$ message to any priest.

Receive $NextBallot(b)$ **Message**

If $b \geq nextBal[p]$ then

– Set *nextBal*[p] to b.

Send *LastVote* **Message**

Enabled whenever $nextBal[p] > prevBal[p]$.

– Send a $LastVote(nextBal[p], v)$ message to priest $owner(nextBal[p])$, where $v_{pst} = p$, $v_{bal} = prevBal[p]$, and $v_{dec} = prevDec[p]$.

Receive *LastVote*(b, v) **Message**

If $b = lastTried[p]$ and $status[p] = trying$, then

– Set $prevVotes[p]$ to the union of its original value and $\{v\}$.

Start Polling Majority Set Q

Enabled when $status[p] = trying$ and $Q \subseteq \{v_{pst} : v \in prevVotes[p]\}$, where Q is a majority set.

– Set $status[p]$ to $polling$.

– Set $quorum[p]$ to Q.

– Set $voters[p]$ to $\emptyset$.

– Set $decree[p]$ to a decree d chosen as follows: Let v be the maximum element of $prevVotes[p]$. If $v_{bal} \neq -\infty$ then $d = v_{dec}$, else d can equal any decree.

– Set $\mathcal{B}$ to the union of its former value and $\{B\}$, where $B_{dec} = d$, $B_{qrm} = Q$, $B_{vot} = \emptyset$, and $B_{bal} = lastTried[p]$.

Send *BeginBallot* **Message**

Enabled when $status[p] = polling$.

– Send a $BeginBallot(lastTried[p],\ decree[p])$ message to any priest in $quorum[p]$.

Receive *BeginBallot*$(b,\ d)$ **Message**

If $b = nextBal[p] > prevBal[p]$ then

– Set $prevBal[p]$ to b.

– Set $prevDec[p]$ to d.

– If there is a ballot B in $\mathcal{B}$ with $B_{bal} = b$ [there will be], then choose any such B [there will be only one] and let the new value of $\mathcal{B}$ be obtained from its old value by setting B_{vot} equal to the union of its old value and $\{p\}$.

Send *Voted* **Message**

Enabled whenever $prevBal[p] \neq -\infty$.

– Send a $Voted(prevBal[p],\ p)$ message to $owner(prevBal[p])$.

Receive *Voted*$(b,\ q)$ **Message**

If $b = lastTried[p]$ and $status[p] = polling$, then

– Set $voters[p]$ to the union of its old value and $\{q\}$

Succeed

Enabled whenever $status[p] = polling$, $quorum[p] \subseteq voters[p]$, and $outcome[p] =$ BLANK.

– Set $outcome[p]$ to $decree[p]$.

Send *Success* **Message**

Enabled whenever $outcome[p] \neq$ BLANK.

– Send a $Success(outcome[p])$ message to any priest.

Receive *Success*(d) **Message**

If $outcome[p] =$ BLANK, then

– Set $outcome[p]$ to d.

This algorithm is an abstract description of the real protocol performed by Paxon priests. Do the algorithm's actions accurately model the actions of the real priests? There were three kinds of actions that a priest could perform "atomically": receiving a message, writing a note or ledger entry, and sending a message. Each of these is represented by a single action of the algorithm, except that **Receive** actions both receive a message and set a variable. We can pretend that the receipt of a message occurred when a priest acted upon the message; if he left the Chamber before acting upon it, then we can pretend that the message was never received. Since this pretense does not affect the consistency condition, we can infer the consistency of the basic Synod protocol from the consistency of the algorithm.

A2 Proof of Consistency

To prove the consistency condition, it is necessary to show that whenever $outcome[p]$ and $outcome[q]$ are both different from BLANK, they are equal. A rigorous correctness proof requires a complete description of the algorithm. The description given above is almost complete. Missing is a variable $\mathcal{M}$ whose value is the multiset of all messages in transit.[15] Each **Send** action adds a message to this multiset and each **Receive** action removes one. Also needed are actions to represent the loss and duplication of messages, as well as a **Forget** action that represents a priest losing his slip of paper.

With these additions, we get an algorithm that defines a set of possible behaviors, in which each change of state corresponds to one of the allowed actions. The Paxons proved correctness by finding a predicate I such that

(1) I is true initially.
(2) I implies the desired correctness condition.
(3) Each allowed action leaves I true.

The predicate I was written as a conjunction $I1 \wedge \ldots \wedge I7$, where $I1$–$I5$ were in turn the conjunction of predicates $I1(p)$–$I5(p)$ for all priests p. Although most variables are mentioned in several of the conjuncts, each variable except $status[p]$ is naturally associated with one conjunct, and each conjunct can be thought of as a constraint on its associated variables. The definitions of the individual conjuncts of I are given below, where a list of items marked by $\wedge$ symbols denotes the conjunction of those items. The variables associated with a conjunct are listed in bracketed comments.

$I1(p) \triangleq$ [Associated variable: $outcome[p]$]

$$(outcome[p] \neq \text{BLANK}) \Rightarrow \exists B \in \mathcal{B} : (B_{qrm} \subseteq B_{vot}) \wedge (B_{dec} = outcome[p])$$

$I2(p) \triangleq$ [Associated variable: $lastTried[p]$]

$$\begin{aligned}
&\wedge\ owner(lastTried[p]) = p \\
&\wedge\ \forall B \in \mathcal{B} : (owner(B_{bal}) = p) \Rightarrow \\
&\qquad\qquad \wedge\ B_{bal} \leq lastTried[p] \\
&\qquad\qquad \wedge\ (status[p] = trying) \Rightarrow (B_{bal} < lastTried[p])
\end{aligned}$$

[15] A multiset is a set that may contain multiple copies of the same element.

$I3(p) \triangleq$ [Associated variables: $prevBal[p]$, $prevDec[p]$, $nextBal[p]$]

$$\begin{aligned}&\wedge\ prevBal[p] = MaxVote(\infty, p, \mathcal{B})_{bal}\\&\wedge\ prevDec[p] = MaxVote(\infty, p, \mathcal{B})_{dec}\\&\wedge\ nextBal[p] \geq prevBal[p]\end{aligned}$$

$I4(p) \triangleq$ [Associated variable: $prevVotes[p]$]

$$\begin{aligned}&(status[p] \neq idle) \Rightarrow\\&\qquad \forall v \in prevVotes[p] : \wedge\ v = MaxVote(lastTried[p], v_{pst}, \mathcal{B})\\&\qquad\qquad\qquad\qquad\qquad\quad \wedge\ nextBal[v_{pst}] \geq lastTried[p]\end{aligned}$$

$I5(p) \triangleq$ [Associated variables: $quorum[p]$, $voters[p]$, $decree[p]$]

$$\begin{aligned}&(status[p] = polling) \Rightarrow\\&\qquad \wedge\ quorum[p] \subseteq \{v_{pst} : v \in prevVotes[p]\}\\&\qquad \wedge\ \exists B \in \mathcal{B} : \wedge\ quorum[p] = B_{qrm}\\&\qquad\qquad\qquad\quad \wedge\ decree[p] = B_{dec}\\&\qquad\qquad\qquad\quad \wedge\ voters[p] \subseteq B_{vot}\\&\qquad\qquad\qquad\quad \wedge\ lastTried[p] = B_{bal}\end{aligned}$$

$I6 \triangleq$ [Associated variable: $\mathcal{B}$]

$$\begin{aligned}&\wedge\ B1(\mathcal{B}) \wedge B2(\mathcal{B}) \wedge B3(\mathcal{B})\\&\wedge\ \forall B \in \mathcal{B} : B_{qrm} \text{ is a majority set}\end{aligned}$$

$I7 \triangleq$ [Associated variable: $\mathcal{M}$]

$$\begin{aligned}&\wedge\ \forall NextBallot(b) \in \mathcal{M} : (b \leq lastTried[owner(b)])\\&\wedge\ \forall LastVote(b, v) \in \mathcal{M} : \wedge\ v = MaxVote(b, v_{pst}, \mathcal{B})\\&\qquad\qquad\qquad\qquad\qquad\quad \wedge\ nextBal[v_{pst}] \geq b\\&\wedge\ \forall BeginBallot(b, d) \in \mathcal{M} : \exists B \in \mathcal{B} : (B_{bal} = b) \wedge (B_{dec} = d)\\&\wedge\ \forall Voted(b, p) \in \mathcal{M} : \exists B \in \mathcal{B} : (B_{bal} = b) \wedge (p \in B_{vot})\\&\wedge\ \forall Success(d) \in \mathcal{M} : \exists p : outcome[p] = d \neq \text{BLANK}\end{aligned}$$

The Paxons had to prove that I satisfies the three conditions given above. The first condition, that I holds initially, requires checking that each conjunct is true for the initial values of all the variables. While not stated explicitly, these initial values can be inferred from the variables' descriptions, and checking the first condition is straightforward. The second condition, that I implies consistency, follows from $I1$, the first conjunct of $I6$, and Theorem 1. The hard part was proving the third condition, the invariance of I, which meant proving that I is left true by every action. This condition is proved by showing that, for each conjunct of I, executing any action when I is true leaves that conjunct true. The proofs are sketched below.

$I1(p)$ $\mathcal{B}$ is changed only by adding a new ballot or adding a new priest to B_{vot} for some $B \in \mathcal{B}$, neither of which can falsify $I1(p)$. The value of $outcome[p]$ is changed only by the **Succeed** and **Receive** *Success* **Message** actions. The enabling condition and $I5(p)$ imply that $I1(p)$ is left true by the **Succeed** action. The enabling condition, $I1(p)$, and the last conjunct of $I7$ imply that $I1(p)$ is left true by the **Receive** *Success* **Message** action.

$I2(p)$ This conjunct depends only on $lastTried[p]$, $status[p]$, and $\mathcal{B}$. Only the **Try New Ballot** action changes $lastTried[p]$, and only that action can set $status[p]$ to $trying$. Since the action increases $lastTried[p]$ to a value b with $owner(b) = p$, it leaves $I2(p)$ true. A completely new element is added to $\mathcal{B}$ only by a **Start Polling** action; the first conjunct of $I2(p)$ and the specification of the action imply that adding this new element does not falsify the second conjunct of $I2(p)$. The only other way $\mathcal{B}$ is changed is by adding a new priest to B_{vot} for some $B \in \mathcal{B}$, which does not affect $I2(p)$.

$I3(p)$ Since votes are never removed from $\mathcal{B}$, the only action that can change $MaxVote(\infty, p, \mathcal{B})$ is one that adds to $\mathcal{B}$ a vote cast by p. Only a **Receive** *BeginBallot* **Message** action can do that, and only that action changes $prevBal[p]$ and $prevDec[p]$. The *BeginBallot* conjunct of $I7$ implies that this action actually does add a vote to $\mathcal{B}$, and $B1(\mathcal{B})$ (the first conjunct of $I6$) implies that there is only one ballot to which the vote can be added. The enabling condition, the assumption that $I3(p)$ holds before executing the action, and the definition of *MaxVote* then imply that the action leaves the first two conjuncts of $I3(p)$ true. The third conjunct is left true because $prevBal[p]$ is changed only by setting it to $nextBal[p]$, and $nextBal[p]$ is never decreased.

$I4(p)$ This conjunct depends only upon the values of $status[p]$, $prevVotes[p]$, $lastTried[p]$, $nextBal[q]$ for some priests q, and $\mathcal{B}$. The value of $status[p]$ is changed from $idle$ to not $idle$ only by a **Try New Ballot** action, which sets $prevVotes[p]$ to $\emptyset$, making $I4(p)$ vacuously true. The only other actions that change $prevVotes[p]$ are the **Forget** action, which leaves $I4(p)$ true because it sets $status[p]$ to $idle$, and the **Receive** *LastVote* **Message** action. It follows from the enabling condition and the *LastVote* conjunct of $I7$ that the **Receive** *LastVote* **Message** action preserves $I4(p)$. The value of $lastTried[p]$ is changed only by the **Try New Ballot** action, which leaves $I4(p)$ true because it sets $status[p]$ to $trying$. The value of $nextBal[q]$ can only increase, which cannot make $I4(p)$ false. Finally, $MaxVote(lastTried[p], v_{pst}, \mathcal{B})$ can be changed only if v_{pst} is added to B_{vot} for some $B \in \mathcal{B}$ with $B_{bal} < lastTried[p]$. But v_{pst} is added to B_{vot} (by a **Receive** *BeginBallot* **Message** action) only if $nextBal[v_{pst}] = B_{bal}$, in which case $I4(p)$ implies that $B_{bal} \geq lastTried[p]$.

$I5(p)$ The value of $status[p]$ is set to $polling$ only by the **Start Polling** action. This action's enabling condition guarantees that the first conjunct becomes true, and it adds the ballot to $\mathcal{B}$ that makes the second conjunct true. No other action changes $quorum[p]$, $decree[p]$, or $lastTried[p]$ while leaving $status[p]$ equal to $polling$. The value of $prevVotes[p]$ cannot be changed while $status[p] = polling$, and $\mathcal{B}$ is changed only by adding new elements or by adding a new priest to B_{vot}. The only remaining possibility for falsifying $I5(p)$ is the addition of a new element to $voters[p]$ by the **Receive** *Voted* **Message** action. The *Voted* conjunct of $I7$, $B1(\mathcal{B})$ (the first conjunct of $I6$), and the action's enabling condition imply that the element added to $voters[p]$ is in B_{vot}, where B is the ballot whose existence is asserted in $I5(p)$.

$I6$ Since B_{bal} and B_{qrm} are never changed for any $B \in \mathcal{B}$, the only way $B1(\mathcal{B})$, $B2(\mathcal{B})$, and the second conjunct of $I6$ can be falsified is by adding a new ballot to

$\mathcal{B}$, which is done only by the **Start Polling Majority Set** Q action when $status[p]$ equals *trying*. It follows from the second conjunct of $I2(p)$ that this action leaves $B1(\mathcal{B})$ true; and the assertion, in the enabling condition, that Q is a majority set implies that the action leaves $B2(\mathcal{B})$ and the second conjunct of $I6$ true. There are two possible ways of falsifying $B3(\mathcal{B})$: changing $MaxVote(B_{bal}, B_{qrm}, \mathcal{B})$ by adding a new vote to $\mathcal{B}$, and adding a new ballot to $\mathcal{B}$. A new vote is added only by the **Receive** *BeginBallot* **Message** action, and $I3(p)$ implies that the action adds a vote later than any other vote cast by p in $\mathcal{B}$, so it cannot change $MaxVote(B_{bal}, B_{qrm}, \mathcal{B})$ for any B in $\mathcal{B}$. Conjunct $I4(p)$ implies that the new ballot added by the **Start Polling** action does not falsify $B3(\mathcal{B})$.

$I7$ $I7$ can be falsified either by adding a new message to $\mathcal{M}$ or by changing the value of another variable on which $I7$ depends. Since $lastTried[p]$ and $nextBal[p]$ are never decreased, changing them cannot make $I7$ false. Since $outcome[p]$ is never changed if its value is not BLANK, changing it cannot falsify $I7$. Since $\mathcal{B}$ is changed only by adding ballots and adding votes, the only change to it that can make $I7$ false is the addition of a vote by v_{pst} that makes the $LastVote(b, v)$ conjunct false by changing $MaxVote(b, v_{pst}, \mathcal{B})$. This can happen only if v_{pst} votes in a ballot B with $B_{bal} < b$. But v_{pst} can vote only in ballot number $nextBal[v_{pst}]$, and the assumption that this conjunct holds initially implies that $nextBal[v_{pst}] \geq b$. Therefore, we need check only that every message that is sent satisfies the condition in the appropriate conjunct of $I7$.

NextBallot: Follows from the definition of the **Send** *NextBallot* **Message** action and the first conjunct of $I2(p)$.

LastVote: The enabling condition of the **Send** *LastVote* **Message** action and $I3(p)$ imply that $MaxVote(nextBal[p], p, \mathcal{B}) = MaxVote(\infty, p, \mathcal{B})$, from which it follows that the *LastVote* message sent by the action satisfies the condition in $I7$.

BeginBallot: Follows from $I5(p)$ and the definition of the **Send** *BeginBallot* **Message** action.

Voted: Follows from $I3(p)$, the definition of *MaxVote*, and the definition of the **Send** *Voted* **Message** action.

Success: Follows from the definition of **Send** *Success* **Message**.

ACKNOWLEDGMENTS

Daniel Duchamp pointed out to me the need for a new state-machine implementation. Discussions with Martín Abadi, Andy Hisgen, Tim Mann, and Garret Swart led me to Paxos. *Λεωνίδας Γκίμπας* provided invaluable assistance with the Paxon dialect.

References

Bernstein, P. A., Hadzilacos, V., and Goodman, N. 1987. *Concurrency Control and Recovery in Database Systems.* Addison-Wesley, Reading, Massachusetts.

De Prisco, R., Lampson, B., and Lynch, N. 1997. Revisiting the Paxos algorithm. In M. Mavronicolas and P. Tsigas (Eds.), *Proceedings of the 11th International Workshop on Distributed Algorithms (WDAG 97)*, Volume 1320 of *Lecture Notes in Computer*

Science, Saarbruken, Germany, pp. 111–125. Springer-Verlag.

Dijkstra, E. W. 1974. Self-stabilizing systems in spite of distributed control. *Commun. ACM 17,* 11 (Nov.), 643–644.

Dwork, C., Lynch, N., and Stockmeyer, L. 1988. Consensus in the presence of partial synchrony. *Journal of the ACM 35,* 2 (April), 288–323.

Fekete, A., Lynch, N., and Shvartsman, A. 1997. Specifying and using a partitionable group communication service. In *Proceedings of the Sixteenth Annual ACM Symposium on Principles of Distributed Computing*, pp. 53–62. ACM Press.

Fischer, M. J., Lynch, N., and Paterson, M. S. 1985. Impossibility of distributed consensus with one faulty process. *Journal of the ACM 32,* 2 (April), 374–382.

Gray, C. G. and Cheriton, D. R. 1989. Leases: An efficient fault-toerant mechanism for distributed file cache consistency. In *Proceedings of the Twelfth ACM Symposium on Operating Systems Principles*, New York, pp. 202–210. ACM.

Keidar, I. and Dolev, D. 1996. Efficient message ordering in dynamic networks. In *Proceedings of the* 15$^{\text{th}}$ *Annual ACM Symposium on Principles of Distributed Computing*. ACM.

Ladin, R., Liskov, B., Shrira, L., and Ghemawat, S. 1992. Providing high availability using lazy replication. *ACM Transactions on Computer Systems 10,* 4 (Nov.), 360–391.

Lamport, L. 1978. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM 21,* 7 (July), 558–565.

Lamport, L. 1984. Using time instead of timeout for fault-tolerant distributed systems. *ACM Trans. on Programm. Lang. Syst. 6,* 2 (April), 254–280.

Lampson, B. W. 1996. How to build a highly available system using consensus. In O. Babaoglu and K. Marzullo (Eds.), *Distributed Algorithms*, Volume 1151 of *Lecture Notes in Computer Science*, Berlin, pp. 1–17. Springer-Verlag.

Oki, B. M. and Liskov, B. H. 1988. Viewstamped replication: A new primary copy method to support highly-available distributed systems. In *Proceedings of the Seventh Annual ACM Symposium on Principles of Distributed Computing*, pp. 8–17. ACM Press.

Schneider, F. B. 1990. Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Computing Surveys 22,* 4 (Dec.), 299–319.

Skeen, M. D. 1982. Crash recovery in a distributed database system. Ph. D. thesis, University of California, Berkeley.

Received January 1990; Accepted March 1998