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 $b = \mathit{lastTried}[p]$ and $\mathit{status}[p] = \mathit{trying}$, then

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

Start Polling Majority Set Q

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

- Set $\mathit{status}[p]$ to *polling*.
- Set $\mathit{quorum}[p]$ to Q.
- Set $\mathit{voters}[p]$ to $\emptyset$.
- Set $\mathit{decree}[p]$ to a decree d chosen as follows: Let v be the maximum element of $\mathit{prevVotes}[p]$. 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 $B_{bal} = \mathit{lastTried}[p]$.

Send *BeginBallot* **Message**

Enabled when $\mathit{status}[p] = \mathit{polling}$.

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

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

If $b = \mathit{nextBal}[p] > \mathit{prevBal}[p]$ then

- Set $\mathit{prevBal}[p]$ to b.
- Set $\mathit{prevDec}[p]$ 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 $\mathit{prevBal}[p] \neq -\infty$.

- Send a $\mathit{Voted}(\mathit{prevBal}[p],\ p)$ message to $\mathit{owner}(\mathit{prevBal}[p])$.

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

If $b = \mathit{lastTried}[p]$ and $\mathit{status}[p] = \mathit{polling}$, then

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

Succeed

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

- Set $\mathit{outcome}[p]$ to $\mathit{decree}[p]$.

Send *Success* **Message**

Enabled whenever $\mathit{outcome}[p] \neq$ BLANK.

- Send a $\mathit{Success}(\mathit{outcome}[p])$ message to any priest.

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

If $\mathit{outcome}[p] =$ 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: $prevBal[p]$, $prevDec[p]$, $nextBal[p]$]

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

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

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

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

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

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

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

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

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

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**.