Question 1 (10 points) YES. If both generals know the total number of messages that will be lost, the 2 Generals Problem can be solved.

Solution (assuming a finite number of messages will be lost):
Let \( L \) be the number of messages that will be lost. Each general sends \( L + 1 \) messages to the other; the contents of each message is the general’s vote. Then, General A will receive at least one message from General B, and General B will receive at least one message from General A. Each general decides on the logical AND of its vote and the other general’s vote (logical OR would also work).

If an infinite number of messages will be lost, the generals can each decide 0.

Question 2 (10 points)
1. YES. 2PC does guarantee agreement if the timeout is too small. The only way agreement will be violated is if some process decides ‘abort’ and another process decides ‘commit’. A process will only decide ‘commit’ if the coordinator receives a vote to commit from every participant, and then sends the ‘commit’ decision to the participants. Suppose this is the case. Then, a participant will either receive the ‘commit’ decision from the coordinator and decide ‘commit’, or it will time out waiting for the decision and execute the cooperative termination protocol. In the cooperative termination protocol, the participant will either learn of the ‘commit’ decision from another participant, or it will remain in the uncertain state. It will never decide ‘abort’. Thus, the system will not violate agreement.

2. NO. 2PC does not guarantee validity of the timeout is too small. A counterexample is as follows. Suppose the coordinator sends the vote request to all participants, and these messages are received before the timeout. All generals respond with ‘commit’, but the coordinator does not receive one of these responses before the timeout. The coordinator thus decides ‘abort’ and sends the decision to the participants. At least one participant receives the decision before the timeout and decides ‘abort’. Thus, all participants voted commit (and no process failed), but the decision was ‘abort’.

Question 3 (10 points)
1. YES. The set \( S = \{ (1, \text{“Skittles”}), (2, \text{“Skittles”}), (\bot, \bot) \} \) could result from a correct execution of the Synod Algorithm. An example execution is as follows.

   1. Proposer \( P_1 \) sends prepare(1). Acceptors \( A_1, A_2, \) and \( A_3 \) receive the messages and respond with promise(\( \bot, \bot \)). \( A_4 \) and \( A_5 \) do not receive the prepare message.

   2. Proposer \( P_1 \) sends accept(1, “Skittles”). Acceptors \( A_1, A_2, \) and \( A_3 \) receive the accept message and accept the proposal. All other messages are lost. Then, \( P_1 \) crashes.

   3. Proposer \( P_2 \) sends prepare(2). Acceptors \( A_1 \) and \( A_3 \) receives the message and respond with promise(1, “Skittles”). Acceptor \( A_4 \) receives the message and responds with promise(\( \bot, \bot \)). \( A_3 \) and \( A_5 \) do not receive the prepare message.

   4. Proposer \( P_2 \) sends accept(2, “Skittles”). Acceptors \( A_2 \) and \( A_4 \) receive the message. All other messages are lost. Then \( P_2 \) crashes.

   5. Proposer \( p \) sends prepare(6). Acceptors \( A_1, A_2, \) and \( A_5 \) respond, and their responses constitute the set \( S \).

In this execution, proposal (1, “Skittles”) is chosen.

2. NO. The set \( S = \{ (3, \text{“Raisins”}), (\bot, \bot), (7, \text{“Snickers”}) \} \) could not result from a correct execution of the Synod algorithm. The acceptor with \( (accNum, accVal) = (7, \text{“Snickers”}) \) would not respond with a promise to a proposal with number 6, since \( 6 < 7 \).

3. YES. The set \( S = \{ (1, \text{“Twix”}), (3, \text{“Twizzlers”}), (5, \text{“Starburst”}) \} \) could result from a correct execution of the Synod Algorithm. An example execution is as follows.

   1. Proposer \( P_1 \) sends prepare(1). Acceptors \( A_1 \) and \( A_2 \) and \( A_3 \) receive the message and respond with promise(\( \bot, \bot \)). \( A_4 \) and \( A_5 \) do not receive the prepare message.
2. \(P_1\) sends \(accept(1, \text{"Twix"})\). Acceptor \(A_1\) receives the message and accepts the proposal. All other \(accept\) messages are lost. Then \(P_1\) fails.

3. Proposer \(P_2\) sends \(prepare(3)\). Acceptors \(A_2\), \(A_3\), and \(A_4\) receive the message and respond with \(promise(\bot, \bot)\). \(A_1\) and \(A_5\) do not receive the prepare message.

4. \(P_2\) sends \(accept(3, \text{"Twizzlers"})\). Acceptor \(A_2\) receives the message and accepts the proposal. All other \(accept\) messages are lost. Then \(P_2\) fails.

5. Proposer \(P_3\) sends \(prepare(5)\). Acceptors \(A_3\), \(A_4\), and \(A_5\) receive the message and respond with \(promise(\bot, \bot)\). \(A_1\) and \(A_2\) do not receive the prepare message.

6. Proposer \(P_3\) sends \(accept(5, \text{"Starburst"})\). Acceptor \(A_3\) receives the message and accepts the proposal. All other \(accept\) messages are lost. Then \(P_3\) fails.

7. Proposer \(p\) sends \(prepare(6)\). Acceptors \(A_1\), \(A_2\), and \(A_3\) respond, and their responses constitute the set \(S\).