ANSWER 1

In the given scenario it is not possible to devise a protocol, which will enable communication using bounded sequence number. Main problem faced in the given scenario is that of channel not being FIFO. Because of this condition protocol has to accommodate garbled messages and reorder them.

Problem exhibits itself in the form of multiple receptions. It is obvious that bounded sequence numbers means protocol has to use messaging windows as underlying technique. This means that in context with the sequence number same scenario can be generated more than once. Complications arise when a message from window w, is received when receiver is expecting messages from window w+i. In this scenario there is no way with which receiver can deduce that currently received message is from window that is past.

One might tempt to think this as multiple delivery problems; but this is more than that. To understand this, assume that currently Wi window messages are being transferred. In this transmission suppose that the channel reordered messages. Because of this M1…. Mi-1 jumped before Mi. Assuming the worst case, lets say that Mi gets delayed indefinitely but is not lost. After some timeout or unacknowledged messages sender sends Mi again, and in second (+) attempts receiver gets it. Now protocol proceeds to next window with Mi still lingering in the channel. Now in next window Wi+1, Mi reaches receiver before Mi+1. Now for receiver there is no chance to differentiate between Mi and Mi+1. Receiver might deliver all messages from Wi+1 including Mi in that set, which violates the condition of ordered delivery.

ANSWER 2

(a)
Program for Q:

```plaintext
define anticipated: integer
initially integer = 0

do
  (m[seq] is received) && (seq = anticipated) []
    accept(m[seq]);
    anticipated := anticipated +1;
```

adjust buffer;

(m[seq] is received) && (seq ≠ anticipated) ☐
buffer[seq – anticipated] := m[seq];

anticipated message found in buffer ☐
accept message;
anticipated := anticipated +1;
adjust buffer;

od

The phrase “adjust buffer” indicates that we push each element in the buffer one place ahead.

This algorithm needs a buffer size S, where S is the upper bound on ‘seq’. In this case, as the message sequence number is unbounded, the channel capacity can be used as required buffer size.

(b)
Now we have a bounded buffer size (which is 1). We use similar idea, but incorporate an acknowledge-based model on top of it. We also assume that message is tagged with a sequence number. Also we put a timeout mechanism in P to ensure no message is lost. Q, however, will ignore multiple copies of the same message.

Program for P:

define clear: boolean, next: integer
initially next = 0, clear = true,
both message channels and acknowledgement channels are empty

do
clear ☐
   send<m, next>;
clear := false;

<ack, next> is received ☐
   next := next + 1;
   ok := true;

timeout (next) ☐
   send<m, next>;
Program for Q:

define  anticipated: integer
initially  anticipated = 0

do

(<m, s> is received) && (s = anticipated) ¶
accept<m, s>;
send<ack, anticipated>;
anticipated := anticipated + 1;
if (buffer is full) ¶ empty buffer fi

(<m, s> is received) && (s ≠ anticipated) && (buffer is empty) ¶
put <m, s> in the buffer;
send<ack, (anticipated – 1)>

(<m, s> is received) && (s ≠ anticipated) && (buffer is full) ¶
ignore <m,s>

od

ANSWER 3

First, we need to assume that the connectivity of the original graph is at least two because, according to the problem statement, the subgraph should be able to broadcast even if one node fails. So our subgraph should also have a connectivity two.

The minimal subgraph suggested for this is a set of Hamiltonian Cycles. One may use a token passing algorithm where the token carries a briefcase of list of visited nodes in order of visit to find out an Hamiltonian. In case, the graph doesn’t have an Hamiltonian path covering all nodes, we’ll try to find a set of Hamiltonian Cycles mutually sharing at least two nodes so that no node becomes critical. Clearly, the message complexity is O(n).
ANSWER 4

Byzantine Generals Algorithm refers to OM(1) message passing. Let us assume, n>3. We consider the following two cases:

Case 1. One of the Lieutenants is traitor.

In this case, OM(1) message received by each Lieutenant will show contradiction in i-th place if Lieutenant i is a traitor. So the faulty process can be suspected. But the same set of values could have been received if the commander was a traitor. So the traitor cannot be identified without ambiguity.

Case 2. The General is traitor.

The general can send some 1's and some 0's (say 2 1’s and 2 0’s). Then with prior knowledge of OM(1), i.e., processes know that at most one process can be faulty, they can identify without any ambiguity the general as the traitor.

But, if the General sends one 1 and rest 0s (or vice versa) then either the General or the Lieutenant receiving the odd message will be suspected as traitor, but the faulty process cannot be identified without ambiguity.
OM(2)

OM(1)

OM(0)