Example Problem: PigeonExpress!

Ben Bitdiddle cannot believe the high valuations of today s Internet companies, so he is doing a startup, PigeonExpress!.com, which provides high performance networking using pigeons. Ben's reasoning is that it is cheaper to build a network using pigeons than it is to dig up streets to lay new cables. Although there is a standard for transmitting Internet datagrams with avian carriers (see network RFC 1149) it is out of date, and Ben has modernized it.

When transmitting a pigeon, Ben's software prints out a little header on a sticky label and also writes a compact disk (CD) containing the data. Someone sticks the label on the disk and gives it to the pigeon. The header on the label contains the Global Positioning System (GPS) coordinates of the destination and the source (the point where the pigeon is taking off), a type field indicating the kind of message (request or acknowledgement), and a sequence number:

Example Problem: PigeonExpress!

```
struct header {
    GPS src;
    GPS dst;
    int type;
    int seq;
}
```

The CD holds a maximum of 640 Megabytes of data. The pigeon reads the header and delivers the labeled CD to the destination. The header and data are never corrupted and never separated. Even better, for purposes of this problem, computers don't fail. However, pigeons occasionally get lost, in which case they never reach their destination.

To make life tolerable on the pigeon network, Ben designs a simple end-to-end protocol (Ben's End-to-End Protocol, BEEP) to ensure reliable delivery. Suppose that there is a single sender and a single receiver. On the sender's computer, the following code is executed:

Example Problem: PigeonExpress!

```
int sequence;     // a global sequence number
int nextCD;       // the next CD to be sent from CD[]

init_sender() { sequence = 0; }

BEEP(GPS dest, int n, CD[]) {      // send n CDs to dest
    struct header h;
    nextCD = 0;
    h.src = my_GPS;                // set src to my GPS coordinates
    h.dst = dest;                  // set dst to the destination
    h.type = REQUEST;              // this is a request message
    while (nextCD < n) {           // send the CDs
        h.seq = sequence;          // set seq number for this CD
        send pigeon with h, CD[nextCD];   //transmit pigeon
        wait 2,000 seconds;
    }
}
```

Example Problem: PigeonExpress!

Pending and incoming acknowledgements are processed only when the sender is waiting:

```
process_ack(struct header h){
    if(h.seq == sequence){
        sequence = sequence + 1;
        nextCD = nextCD + 1;        // allow next CD to be sent
    }
}
```

The receiver executes the following code:

```
process_request(struct header h, CD) { // process request
    process(CD);          // process the data on the CD
    h.dst = h.src;        //send to where the pigeon came from
    h.src = my_GPS;
    h.seq = h.seq;        //unchanged
    h.type = ACK;         // this is an acknowledgement
    send pigeon with h;  // send an acknowledgement back
}
```

Question 1

If a pigeon travels at 100 meters/second (these are express pigeons!) and pigeons do not get lost, then what is the maximum data rate observed by the caller of BEEP on a 50,000 meter (50 kilometer) long pigeon link? Assume that the processing delay at the sender and receiver are negligible.

Question 1

If a pigeon travels at 100 meters/second (these are express pigeons!) and pigeons do not get lost, then what is the maximum data rate observed by the caller of BEEP on a 50,000 meter (50 kilometer) long pigeon link? Assume that the processing delay at the sender and receiver are negligible.

640 MB / 2000 seconds, or 320 Kilobytes/second

Question 2

- Does at least one copy of each CD make it to the destination, even though some pigeons are lost?

  A. Yes, because *nextCD* and *sequence* are incremented only on the arrival of a matching acknowledgement.

  B. No, since there is no explicit loss-recovery procedure (such as a time-out procedure).

  C. No, since other requests and acknowledgements can get lost.

  D. Yes, since the next acknowledgement will trigger a retransmission.

Question 2

- Does at least one copy of each CD make it to the destination, even though some pigeons are lost?

A. Yes, because *nextCD* and *sequence* are incremented only on the arrival of a matching acknowledgement.

B. No, since there is no explicit loss-recovery procedure (such as a time-out procedure).

C. No, since other requests and acknowledgements can get lost.

D. Yes, since the next acknowledgement will trigger a retransmission.

Question 3

- To guaranteee that each CD arrives at most once, what is required?

A. We must assume that a pigeon for each CD has to arrive eventually.

B. We must assume that acknowledgement pigeons do not get lost and must arrive within 2,000 seconds after the corresponding request pigeon is transmitted.

C. We must assume request pigeons must never get lost.

D. Nothing. The protocol guaranteees at-most-once delivery.

Question 3

- To guaranteee that each CD arrives at most once, what is required?

A. We must assume that a pigeon for each CD has to arrive eventually.

B. We must assume that acknowledgement pigeons do not get lost and must arrive within 2,000 seconds after the corresponding request pigeon is transmitted.

C. We must assume request pigeons must never get lost.

D. Nothing. The protocol guaranteees at-most-once delivery.

Question 4

- Ignoring possible duplicates, what is needed to guarantee that CDs arrive in order?

  A. We must assume that pigeons arrive in the order in which they were sent.

  B. Nothing. The protocol guarantees that CDs arrive in order.

  C. We must assume that request pigeons never get lost.

  D. We must assume that acknowledgement pigeons never get lost.

Question 4

- Ignoring possible duplicates, what is needed to guarantee that CDs arrive in order?

A. We must assume that pigeons arrive in the order in which they were sent.

B. Nothing. The protocol guarantees that CDs arrive in order.

C. We must assume that request pigeons never get lost.

D. We must assume that acknowledgement pigeons never get lost.

Example Problem: PigeonExpress!

To attract more users to PigeonExpress!, Ben improves throughput of the 50-Kilometer long link by using a window-based flow-control scheme. He picks *window* (number of CDs) as the window size and rewrites the code. The code to be executed on the sender's computer is
now as follows:

Example Problem: PigeonExpress!

```
int sequence;       // global sequence number
int nextCD;         // next CD to be sent from CD[]
int window;         // window size

init_sender() { sequence = 0;}

BEEP(GPS dest, int n, CD[]) {           // send n CDs of data
    struct header h;
    int CDsleft, j, t;
    nextCD = 0;
    window = 10;        // initial window size is 10 CDs
    h.src = my_GPS;
    h.dst = dest;
    h.type = REQUEST;
    while (nextCD < n) {
        CDsleft = n - nextCD;       //foo computes how many pigeons to send
        t = foo(CDsleft, window);   // send next t pigeons;
        for (j = 0; j < t; j = j + 1) {
            h.seq = sequence + j;
            send pigeon with h, CD[nextCD + j] to dest
        }
    wait 2,000 seconds
    }
}
```

Example Problem: PigeonExpress!

Pending and incoming acknowledgements are processed only when the sender is waiting:

```
process_ack(struct header h) {
   if (h.seq == sequence) {
      sequence = sequence + 1;
      nextCD = nextCD + 1; // increase nextCD index
   }
}
```

There are no changes to *process_request* that runs on the receiver's computer.

Question 5

- What should **foo** compute in this code fragment?

  A. minimum

  B. maximum.

  C. sum.

  D. absolute difference.

Question 5

- What should **foo** compute in this code fragment?

A. minimum

B. maximum.

C. sum.

D. absolute difference.

Question 6

- Alyssa looks at the code and tells Ben it may lose a CD. Ben is shocked and disappointed. What should Ben change to fix the problem?

  A. Nothing. The protocol is fine; Alyssa is wrong.

  B. Ben should modify process_request to accept and process CDs in the order of their sequence numbers.

  C. Ben should set the window size to the delay * bandwidth product.

  D. Ben should ensure that the sender sends at least one CD after waiting for 2,000 seconds.

Question 6

- Alyssa looks at the code and tells Ben it may lose a CD. Ben is shocked and disappointed. What should Ben change to fix the problem?

A. Nothing. The protocol is fine; Alyssa is wrong.

B. Ben should modify process_request to accept and process CDs in the order of their sequence numbers.

C. Ben should set the window size to the delay * bandwidth product.

D. Ben should ensure that the sender sends at least one CD after waiting for 2,000 seconds.

Question 7

- Assume pigeons do not get lost. Under what assumptions is the observed data rate for the window-based BEEP larger than the observed data rate for the previous BEEP implementation?

  A. The time to process and launch a request pigeon is less than 2,000 seconds;

  B. The sender and receiver can process more than one request every 2,000 seconds;

  C. The receiver can process less than one pigeon every 2,000 seconds;

Question 7

- Assume pigeons do not get lost. Under what assumptions is the observed data rate for the window-based BEEP larger than the observed data rate for the previous BEEP implementation?

A. The time to process and launch a request pigeon is less than 2,000 seconds;

B. The sender and receiver can process more than one request every 2,000 seconds;

C. The receiver can process less than one pigeon every 2,000 seconds;

Example Problem: PigeonExpress!

After the initial success of the PigeonExpress!, the pigeons have to travel farther and farther, and Ben notices that more and more pigeons don t make it to their destinations, because they are running out of food. To solve this problem, Ben calls up a number of his friends in strategic locations and asks each of them to be a hub, where pigeons can reload on food.

To keep the hub design simple, each hub can feed one pigeon per second and each hub has space for 100 pigeons. Pigeons feed in FIFO (first in first out) order at a hub. If a pigeon arrives at a full hub, the pigeon gets lucky and retires from PigeonExpress!. The hubs run a patented protocol to determine the best path that pigeons should travel from the source to the destination.

Question 8

- Which layer in the reference model of Chapter 4 provides functions most similar to the system of hubs?

  A. the end-to-end layer

  B. the network layer

  C. the link layer

  D. network layer and end-to-end layer

  E. the feeding layer

Question 8

- Which layer in the reference model of Chapter 4 provides functions most similar to the system of hubs?

A. the end-to-end layer

B. the network layer

C. the link layer

D. network layer and end-to-end layer

E. the feeding layer

Question 9

- Assume Ben is using the window-based BEEP implementation. What change can Ben make to this BEEP implementation in order to make it respond gracefully to congested hubs?

  A. add a congestion window to BEEP, which starts out with a smallvalue and increases upon the arrival of an acknowledgement;

  B. have process_request delay acknowledgements and have a single pigeon deliver multiple acknowledgements;

  C. use a window size smaller than 100 CDs, since the hub can hold 100 pigeons;

  D. use multiplicative decrease and additive increase for the window size.

Question 9

- Assume Ben is using the window-based BEEP implementation. What change can Ben make to this BEEP implementation in order to make it respond gracefully to congested hubs?

A. add a congestion window to BEEP, which starts out with a smallvalue and increases upon the arrival of an acknowledgement;

B. have process_request delay acknowledgements and have a single pigeon deliver multiple acknowledgements;

C. use a window size smaller than 100 CDs, since the hub can hold 100 pigeons;

D. use multiplicative decrease and additive increase for the window size.