4.3) Assuming that there is no collision, *Pure ALOHA (PA)* will tend to have less delay. This is because *PA* does not ever need to wait to transmit. *Slotted ALOHA (SA)*, on the other hand, will wait for the next available time slice. For applications with extraordinarily low throughput (data is not transmitted very often), this is acceptable. As we all know, the real world is rarely this forgiving.

4.7) Given worst case scenario: assume there are *N* stations, and s is the first one *(N_0)*. If s misses its spot in the contention slot it must wait for the contention slot to come back around. Now assume (worst-case again) that every other station has successfully inserted their respective bit in the contention slot. So after the contention slot rolls through, s must now wait for N-*1* frames to be transmitted before the contention slot comes back through. If each frame is d bits wide, s must wait N-*1* bits for the rest of its missed contention slot AND *((N-1)*d)* bits for all other queued frames to transmit. This gives a worst case wait-time of:

*(N-1)+(N-1)*d*

which can be expressed as:

*(N-1) * (d+1)*

For *N* stations and frame transmit time of *d* bits

4.38)

A → C:

On B1, frame is forwarded to ports 2, 3, 4
When B2 receives that same frame from B1, it is forwarded to ports 1, 2, 3 on B2
B1 now knows A can be accessed via port 1
B2 now knows A can be accessed via port 4

E → F:

Because H1 (repeater) does not maintain LUT/hash table, it repeats E's frame everywhere.
B2 receives frame and broadcasts it on ports 1, 3, 4
B1 receives frame and broadcasts it on ports 1, 2, 3
B2 now knows E can be accessed via port 2

F → E:

H1 repeats F's frame to E and B2, E now has the frame
B2 already knows where E is and so does NOT broadcast frame to any other ports
B2 discards frame
B2 now knows F can be accessed via port 2 (B1 does not know because B2 discarded frame)

G → E

On B2, frame is forwarded to ports 1, 2, 4
On B1, frame is re-forwarded to ports 1, 2, 3
H1 broadcasts frame to E and F; E now has frame
B2 now knows G can be accessed via port 3
B1 now knows G can be accessed via port 4

D → A

B2 forwards frame to port 4
B1 forwards frame to port 1
B2 now knows D can be accessed via port 1
B1 now knows D can be accessed via port 4

B → F

B1 forwards frame to ports 1, 3, 4 (because it never learned where F is)
B2 forwards frame to port 2

B1 now knows B can be accessed via port 2
B2 now knows B can be accessed via port 4

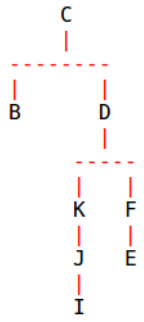
5.6) Using Distance Vector Routing
New routing table for C:

To node:	A	B	C	D	E	F
$Cost_{line}$	11_B	6_B	0_C	3_D	5_E	8_B

5.11)
Reverse tree forwarding:
25 packets broadcast by all routers in network
Sink Tree:
18 packets broadcast by all routers in network (much less than Reverse Tree Forwarding)

5.12)
Following the paths of 5-15(c) to what would be the new node (Z):
I → F → Z
I → J → G → Z
It becomes clear that F would have knowledge of Z well before G, hence F would gain a new end-node and G would ignore it

5.13)
The pruned multicast spanning tree could look like:



5.14)
With TtL set to 1, packet will reach A, D, C
With TtL set to 2, packet will also reach nodes F, G, E
With TtL set to 3, packet will reach H. With this, one of two paths will be created:
1.) B ↔ D ↔ G ↔ H
2.) B ↔ D ↔ F ↔ H
depending on which path had lower latency during discovery. In either case, the target has been found in 3 rounds