Part 1:

For the case of linear Tree: considering just one message transferring from root processor to final processor , one may get simply the following formula for the time required to do this procedure and naming it as “Completion Time” (described in figure.1):

*Completion Time for linear Tree =*

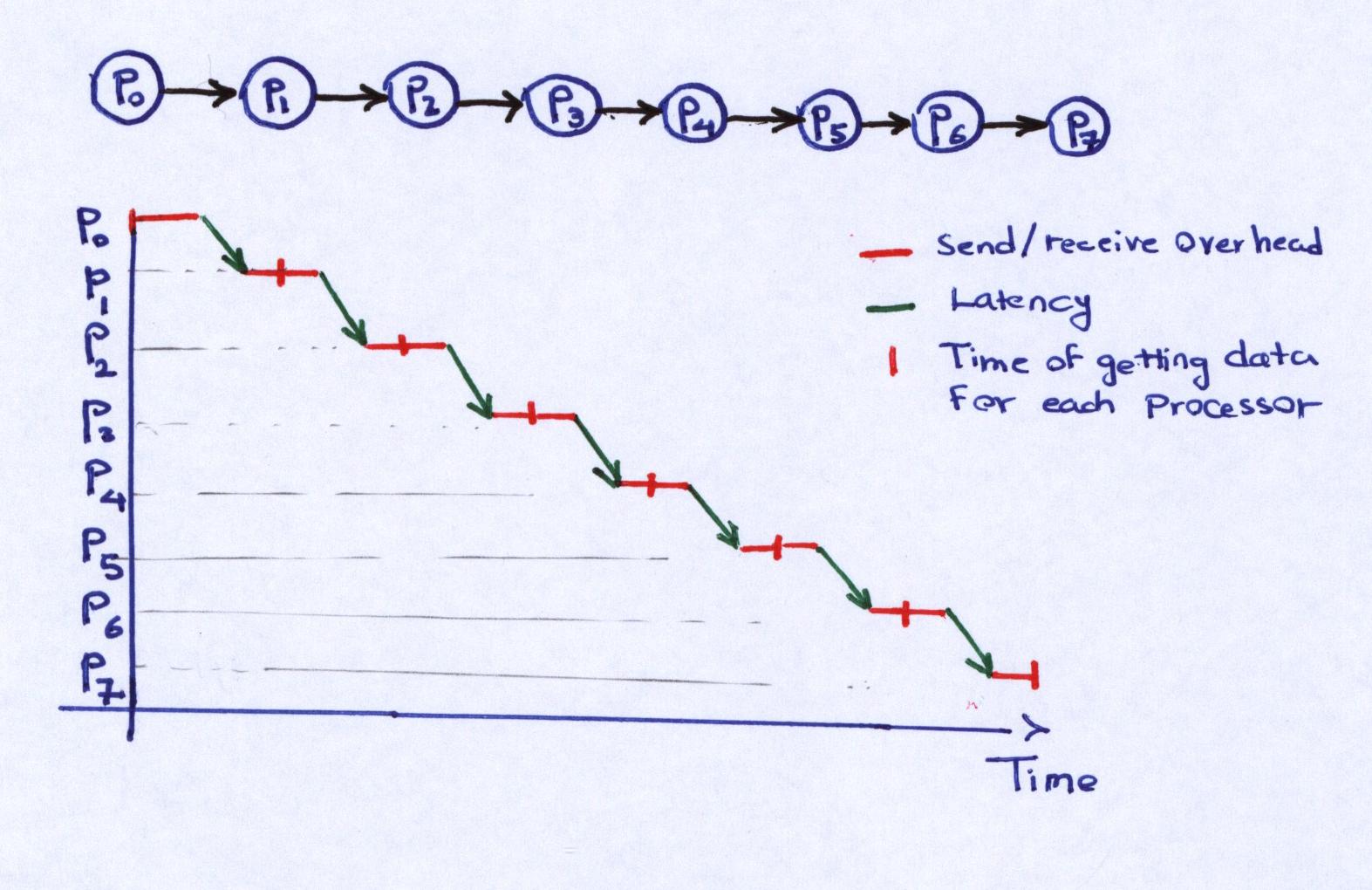


Figure 1. Representation of Linear Tree message sending.

where p is the number of processors, Or is receiving overhead, Os is sending overhead and L is Latency for transmission from one node to the other.

For the case of Binomial Tree: Considering 8 processors, one may have the processors in following pattern as figure.2.

So if we want to derive a formula which relates the number of processors and the stage that we are in, we may take a look at figure.2 and we have:

; Here S0, S1…are the number of stages.

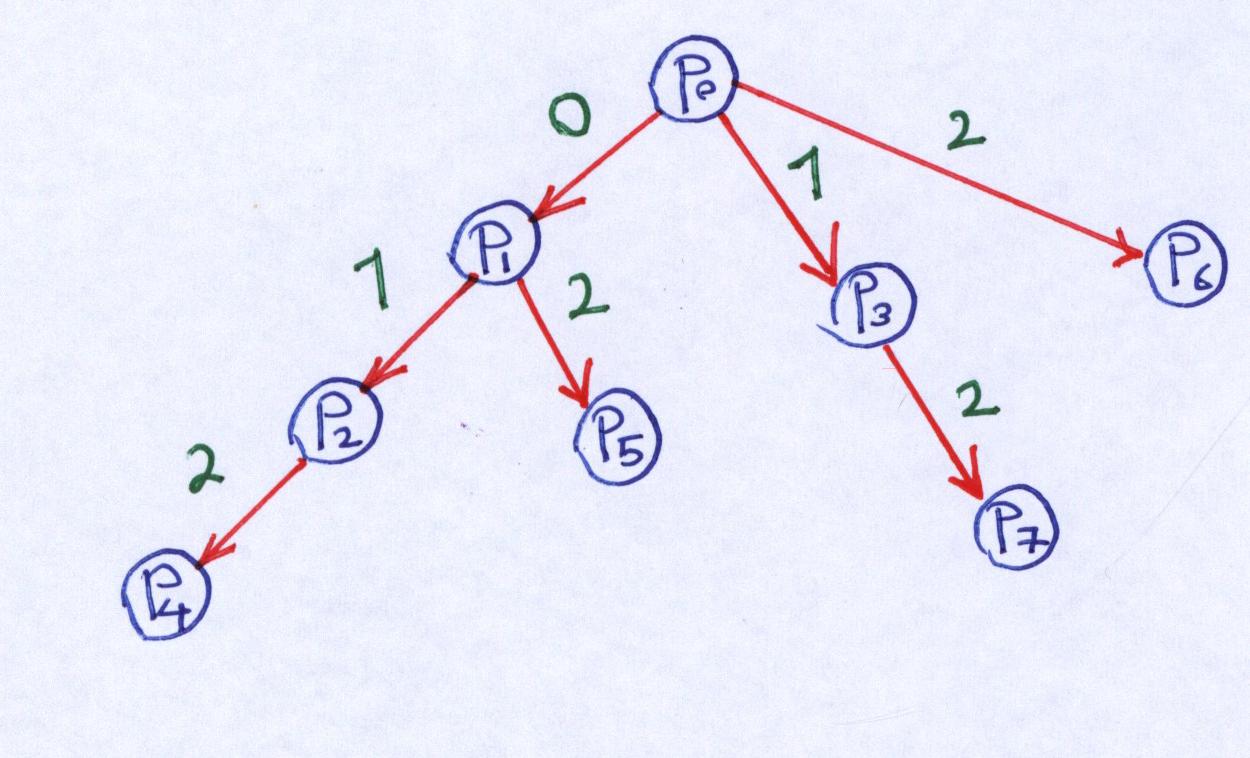


Figure 2. Representation of binomial Tree.

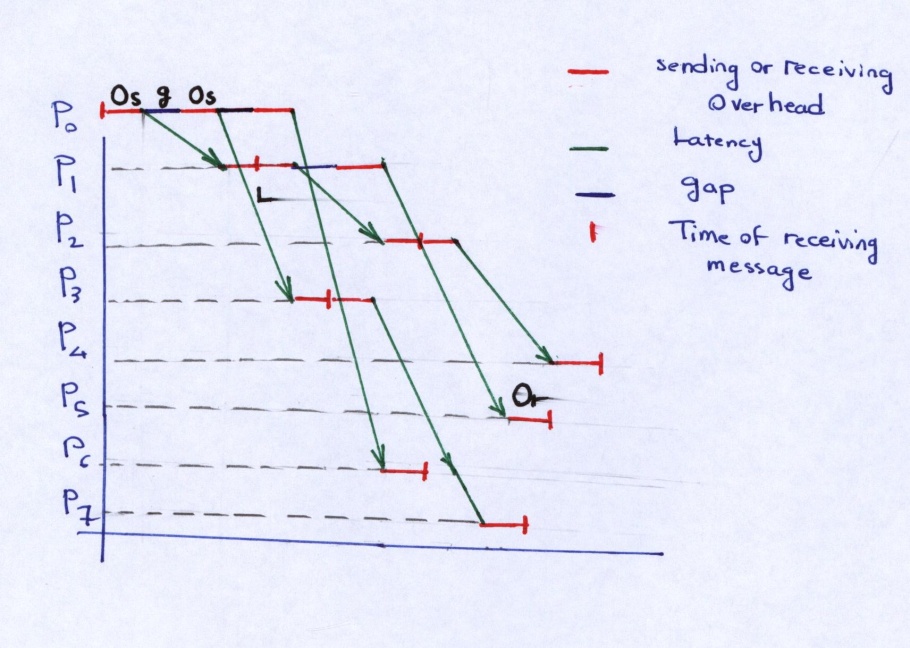


Figure 3. Sending and receiving message in Binomial Tree.

Therefore, one may conclude that p = 2h, where h is the number of steps and p is the number of processors. So, this would be in another word the height of the tree.

For instance, I plot the timing for a message in the above described Binomial Tree (see figure.3):

The final time, i.e. the time at which the specific processors have received the message could be summarized as follows:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Pi | P1 | P2 | P3 | P4 | P5 | P6 | P7 |
| Time | Os+Or+L | 2(Os+Or+L) | 2Os+Or+L+g | 3(Os+Or+L) | 3Os+2Or+2L+g | 3Os+2g+L | 3Os+2Or+g+2L |

In this simple case, I assumed that L is approximately 2-3 times larger than g and O (I don’t know whether it is always the case); therefore, as it is clear in figure.3, the dominating time which would be the overall completion time is for P4, which is the furthest node from the root. The completion time is then the height of the tree multiplied by time required for a message to be transferred between each two machines:

*Completion Time for binomial Tree =*

Table 1. functionality of L(x), g(x) and O(x) for a specific machine.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Machine | L(µs) | g(µs) | o(µs) | P |
| Para station(x~KB) | 50-0.1x | 3+0.112x | 3+0.119x | 4 but I use for 8 |

I do not know their units. I assume that L, g and o is in micro seconds and x is in Kbyte. Therefore, for a message of relatively small size like 1K and for large like 256K we have:

Table 2. The value of L, g and O for short and large size messages, based on Table.1.

|  |  |  |  |
| --- | --- | --- | --- |
| size | L(µs) | g(µs) | o(µs) |
| 1KB | 49.9 | 3.112 | 3.119 |
| 256KB | 24.4 | 31.672 | 33.464 |

With these approximate data, one can obtain the cost of each topology. The following figure compares Linear Tree and Binomial Tree:

Based on this result, the Binomial Tree would be more efficient than Linear Tree, which we somehow expected as the message proceed faster in this topology than Linear Tree. No matter the size of message is, the Binomial Tree is more efficient than Linear Tree.



Figure 4. Broadcasting cost for Linear Tree and Binomial Tree for short size messages.



Figure 5.Broadcasting cost for Linear Tree and Binomial Tree for large messages.

For the case of two binary trees, I have a feeling about how it works. However, I am not pretty sure that I derive the same formulation that is right for this method. So, I skip doing this section for Two Binary Tree.

Part 2: (For Pipelined performance)

First the cases in which the gap between sending each fragment is somehow not considered.

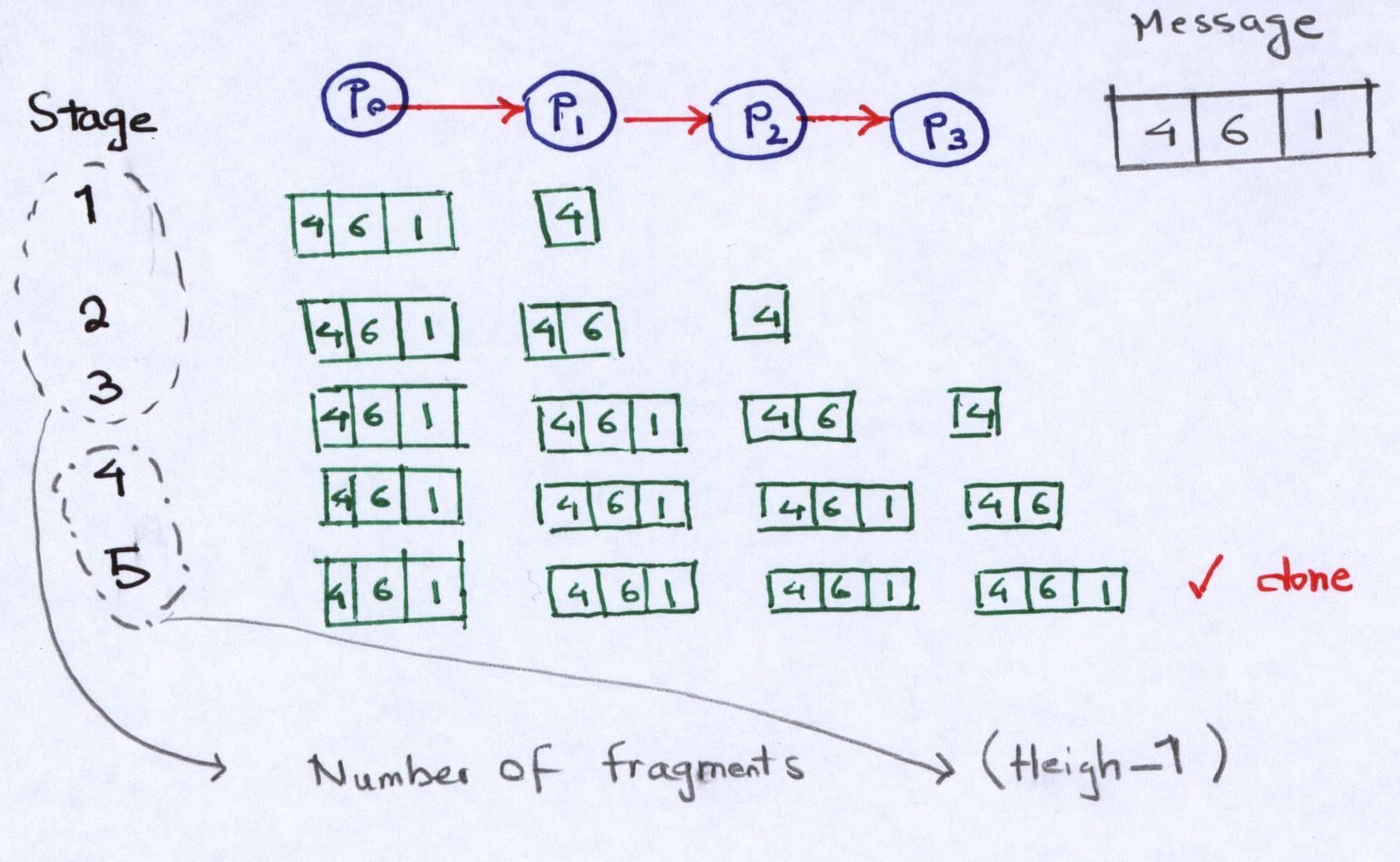
For the case of Linear Tree: if I break the message into k segments, then the time which is needed to transfer the k fragments to the last processor may be calculated by plotting the procedure in Figure.6:

Figure 6. Pipelined propagation in Linear Tree.

It could be observed that there are k stage of communication first which is followed by h-1 stages which complete the job and therefore the last node receives the whole message. So, one may write the completion time as follows:

*Completion Time for pipelined linear Tree =*

The receiving and sending overhead as well as latency are described as a function of (m/k) which is the size of fragment of the message.

For the case of Binomial Tree: I plotted the procedure of propagating message fragment in this system (see figure. 7). Based on my observations, I ended up getting something like:

*Completion Time for pipelined binomial Tree =*

where C is the number of children of root processor.

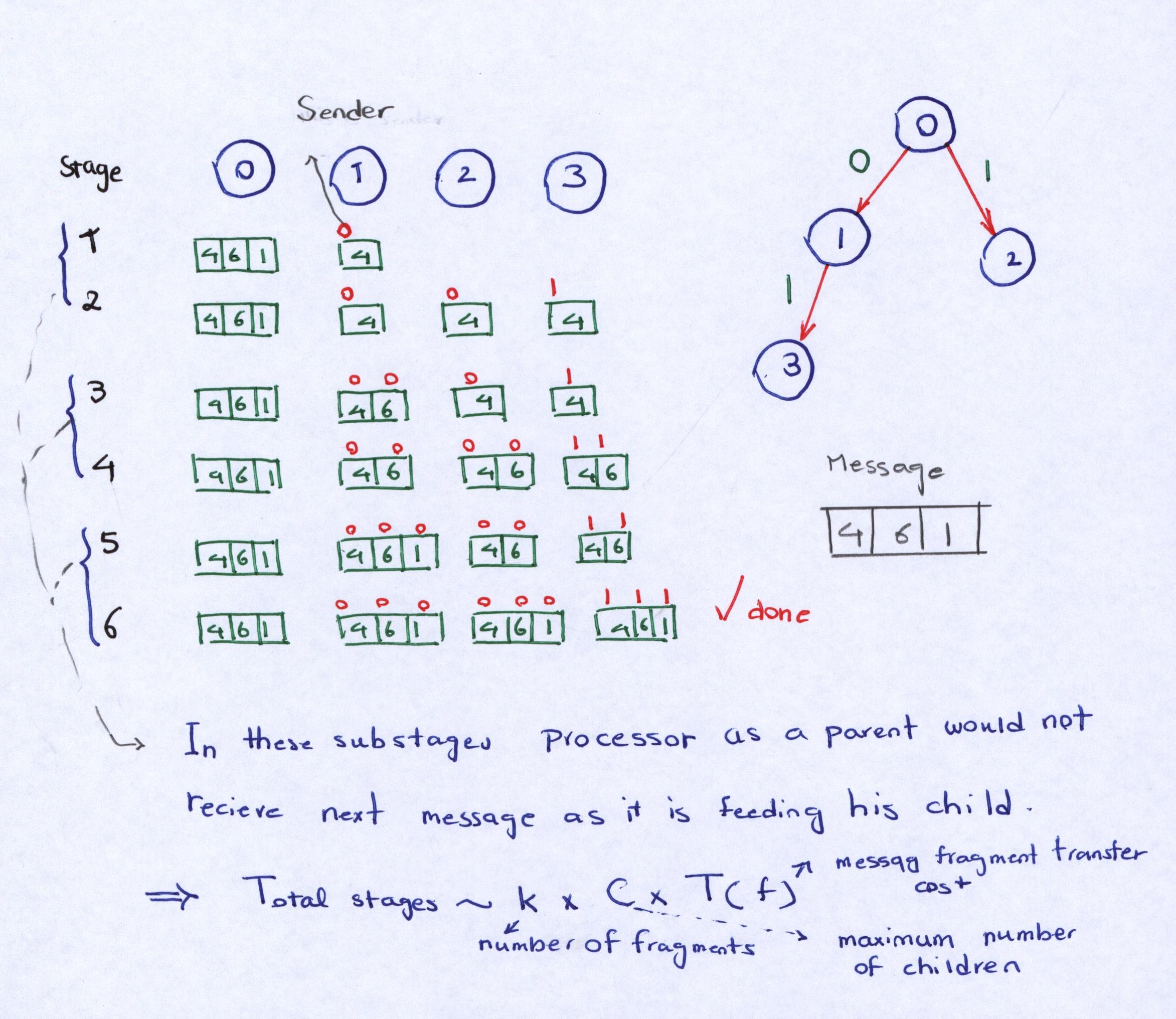


Figure . Pipelined propagation in Binomial Tree.

However as I searched the literature, I found that for general case of a tree, the following formulation hold:

*Completion Time for pipelined binomial Tree in literature =*

where D is the maximum nodal degree of tree (which would be the same as C) and h is the height of the tree. is the startup overhead and is the transmission time per byte. I accept this formula in next discussions.

By accepting this formula, one would say that if the size of the message is large, k would be much greater than h-1 and therefore . If we assume that is approximately , which is true if the overhead for sending k messages of m/k size is approximately the same as overhead of sending message m, then we have:

We know that our ideal condition is the case in which . So, we would increase the performance if we decrease D. In other words, (by simple analysis) Linear Tree is more efficient than Binomial Tree.

To have some more detailed data:

for Linear Tree: h=p-1 and D=1, therefore for large messages; . which would be very close to ideal situation.

For Binomial Three:

However, for other cases with unknown size, when , then Binary Tree would be more efficient than Linear Tree.

For the case of Two Binary Tree : I used the formulation given in the paper as:

*Completion Time for pipelined Two-binary Tree in literature =*

In order to calculate α and β, I used the slides in class and also Table.1:

Now, if I split a message into 4 pieces, then I have 4 messages with m/4 size which I may use all these formula to obtain:

For small messages; between 4 to 10K.



Figure 8. Cost of Broadcasting for three different topologies for small message.

for large messages; between 256 to 500K



Figure 9. Broadcasting cost for three different types for large messages.

Therefore, for both sizes, the performance of Two binary Tree is better than Linear and Linear is better than Binomial.

Second section :( changing number of processors for medium sized message)



Figure 10. Broadcasting cost for Linear and Binary Tree (for small size message).

Then without pipelined method, the performance of Binomial Tree is still better than Linear.



Figure 11. Broadcasting cost for Pipelined Linear and Binary Tree (for small size message).

It is interesting that above a certain number of processors, the binomial would be more efficient than Linear in pipelined topology.

For Large size messages: there was not much change relative to small size message. so I ignored to show them.

I expected that message size had much more effect on trends. Maybe the units are not the same as taken here. So, the results may change partially. Also, the gap may be considered in calculations and there are some formulation in literature for Linear Tree and considering gap but removing the extra contribution of Or and Os.