of T_k. There is also additional time for the *probe* and *ack/reject* messages at each step totalling $2k + 2$ time for each step. Therefore, assuming that the final *BFS* tree will be at most of depth d,

$$\begin{aligned} Time(Synch\_BFS) &= \sum_k Time(Phase_k) \\ &= 2\sum_{k=1}^{d}(k+1) = (d+1)(d+2)/2 = O(d^2). \end{aligned} \quad (5.1)$$

For the message complexity, we need to consider the synchronization messages and tree forming messages separately. In round p, if the tree formed has k nodes, there will be $k - 1$ *round* messages and $k - 1$ *upcast* messages for a total of $2k - 1$ messages providing a maximum number of synchronization messages in a round as $O(n)$. Since the formed BFS tree will have diameter d as the depth, the message complexity for the synchronization process is $O(nd)$. In each round, new edges of the BFS tree will be determined by the *probe* and $ack/nack$ messages, and therefore, the total traversals for discovery of these edges will be at most $2m$. Summation of the synchronization and discovery messages yields:

$$Msg(Synch\_BFS) = \sum_p msg(Phase\ p) = O(nd) + O(m) = O(nd + m). \quad (5.2)$$

□

5.2.2 *Asynchronous BFS Construction*

The second algorithm to build a BFS tree of a graph G is the distributed version of the Bellman–Ford algorithm called *Update_BFS*. We have a single initiator as before, which starts the algorithm by sending the *layer*(l) message that contains its distance to its neighbors as unity. Any node receiving a *layer*(1) message compares the layer value l contained in the message with its known distance to the root, and if the new value is smaller, the sender of the layer message is labeled as the new parent, and the distance is updated to l. Since the new distance to the root will affect all neighbors and other nodes, the *layer*$(l + 1)$ message containing the new distance is sent to all neighbors except the new parent as shown in Algorithm 5.2. It can be seen that this process eventually builds a BFS tree starting from the root. The termination condition would be the traversing of the longest shortest path between any two nodes, which would be the diameter of the graph G.

5.2.2.1 Example

An example is shown in Fig. 5.3 with six nodes numbered $1, \ldots, 6$, where the *layer* message carries the distance, and the time frame it is delivered as *layer(distance,*

Algorithm 5.2 *Update_BFS*

```
int parent ← ∅, my_layer ← ∞, count = 1, d ← diameter of G
set of int childs ← ∅, others ← ∅
message types layer, ack, reject
if i = root then
    send layer(1) to Γ(i)
end if
while count ≤ d do
    receive msg(j)
    case msg(j).type of
        layer(l):  if my_layer > l then              ▷ update distance
                       parent ← j
                       my_layer ← l
                       send ack(l) to j              ▷ inform parent i am child
                       send layer(l+1) to Γ(i)\{j}   ▷ update neighbors
                   else
                       send reject(l) to j           ▷ else reject sender
        ack(l):    childs ← childs ∪ {j}             ▷ include sender in children
        reject(l): others ← others ∪ {j}             ▷ include sender in unrelated
    count ← count + 1
end while
```

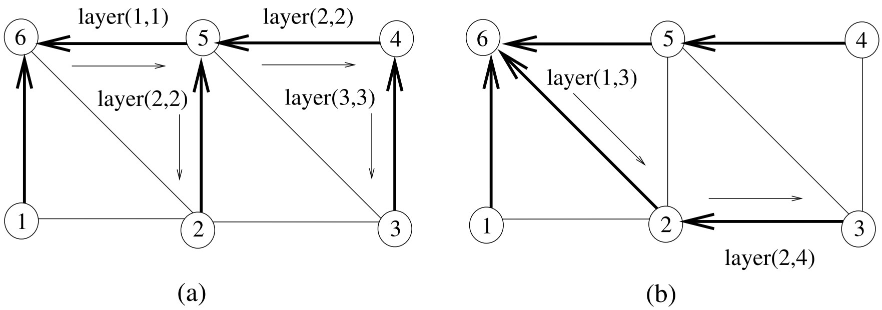

Fig. 5.3 *Update_BFS* execution example

time). Node 6 initiates the algorithm by sending the *layer*(1, 1) message to its one-hop neighbors. Each neighbor node, when it receives this message, compares the distance value in the message to its known distance and assigns its parent to the sender if the new distance is smaller. It can be seen in Fig. 5.3(a) that *layer* message reaches node 2 via node 5 before the direct connection between nodes 6 and 2, resulting in node 2 identifying node 5 as its parent. However, this situation is corrected in (b) when the layer message from node 6 reaches node 2 in the third time frame resulting in node 2 replacing its parent node 5 with node 6 correctly. Similarly, in time frame 4, node 3 replaces its parent node 4 with node 2 to correctly construct the BFS tree rooted at node 6.