Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

FS 2008 Prof. R. Wattenhofer / Dr. F. Kuhn / T. Locher / Y. A. Oswald / C. Lenzen

Principles of Distributed Computing
Exercise 3: Sample Solution

1 License to Kill

a) We use a variant of the Echo Algorithm 3.7. A node (i.e. an agent in the hierarchy) matches up all (except for at most one) of his children. If the node or one of its children is left out, then the node sends a request to "match" upwards in the hierarchy. Otherwise, it sends a "no match" and that subtree is done. We give an asynchronous, uniform algorithm below.

Algorithm 1 Edge-Disjoint Matching

```
wait until received message from all children
while remain ≥ 2 requests (including myself) do
  match any two requests
end while
if exists leftover request then
  send up "match"
else
  send up "no match"
end if
```

When a node v sends a "match" request to its parent u, then the edge $\{u, v\}$ will be used only once since there will be only one request in the subtree rooted at v.

b) Let T be the tree with n nodes. Assuming each message takes at most 1 time unit, then the time complexity of Algorithm 1 is in $\mathrm{O}(depth(T))$ since all the requests travel to the root (and back down if we inform the agents of their assigned partners). We can assume that local computation (matching) is negligible. On each link, there are at most 2 messages: 1 that informs the parent whether a match is needed and optionally 1 more to be informed by the parent of the match partner. So there are a total of at most $2(n-1)$ messages.

2 License to Distribute

a) Again we apply an echo-style algorithm where a node locally balances the documents as much as possible. For each node v we can define

$$balance(v) := have(v) - need(v)$$

where $have(v)$ is the total number of documents and $need(v)$ is the number of nodes in the subtree rooted at v. Each node then computes and sends this balance to its parent. Again the algorithm is asynchronous and uniform. Abstractly we refer to a document as a token. When we gather the balance information from the children, they implicitely also send along any extra tokens they might have.

Algorithm 2 Token Distribution for node v

```
wait until received balance from all children
balance(v) := 0
for each child c do
    balance(v) := balance(v) + balance(c)
end for
balance(v) := balance(v) + tokens(v) - 1
send up balance(v)
if balance(v) < 0 then
    wait to receive needed tokens from parent
end if
redistribute tokens among children
```

b) The time complexity is in $O(depth(T))$ analogous to Exercise 1. If we assume that we can (physically) send all the tokens in one message, then again there is one message upwards for each link and optionally one downwards with the missing tokens. Thus there are at most $2(n-1)$ messages.