Airline Reservation System

Valentin Marcu
An I SCPD
University Politehnica of Bucharest
Email: valentin.marcu@cti.pub.ro

***Abstract*—This articles describes a simple distributed system which permits the clients to book travels that involve multiple flights. Each airport maintains its own collection of servers and is able to wisely route requests and replies between any nodes in the network. All the airports are assigned to geographical zones, which are used by the routing policies, resulting in a smaller routing overhead. In order to avoid broadcasting queries to all its neighbours, each airport forwards them only to neighbours placed within the next-hop zones towards the destination.**

I. Introduction

Large scale distributed systems often need to implement smart routing techniques in order to avoid high traffic throughout the network, especially when large amounts of messages are expected to flow between any nodes, regardless of their geographical positions. Our application implements such a technique, based on the idea that when a message needs to be sent at a distant location, it isn't necessary to know much details about the destination, but only a hint of where that destination is located.

The next section of the paper describes the routing algorithm, including prerequisites (some static information that each airport should have) and some simplified characteristics that we have implemented in our tests. Also, some possible future optimizations are provided in the last section.

II. Implementation

A. Zones of airports

Each airport throughout the world is assigned to a geographical zone, whose dimensions are big enough to reflect a large group of relatively close entities and small enough to maintain a minimum set of common geographical characteristics (for example, a zone can be a small continent like Europe or a homogenous and dense group of airports such as USA). Each airport and zone has an ID; an airport and a zone may have the same ID, but no two airports and no two zones may have the same ID.

The assignment process is done only once and its results are supposed to be already known to all the airports in the network.

B. Prerequisites

During initialization, each server (corresponding to an airport) reads a configuration file, containing static information about the network, such as:

- N - number of airports
- Z - number of zones
- zone[N] - the airport-zone mapping
- neighbours (ArrayList¡Integer¿) - the IDs of the local node's neighbours (towards whom there are direct flights)
- next (ArrayList¡ArrayList¡Integer¿¿) - next hop zones towards all the other zones (multiple zones may be next hops towards a single other zone)

An additional array of Integers (last[N]) is used by each server, having the meaning that last[i] is the neighbour that last forwarded to the local server a message originating from i. This information is used whenever a reply must be forwarded back to the source of a query.

C. Messages

Messages sent throughout the network are of two types: request and reply. Each message contains its source and destination, along with 3 integer fields used for routing requests by the intermediate nodes:

- via - the current node
- next - the neighbour node towards which the message must be forwarded
- TTL - the maximum number of nodes within a path (user defined)

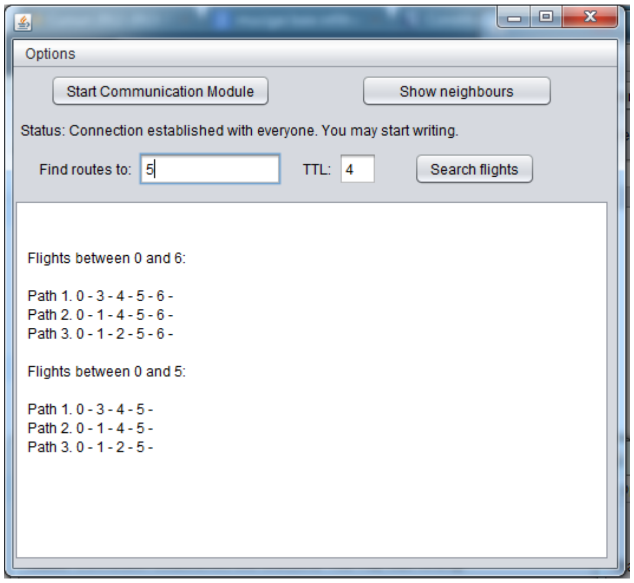
As a message is routed, a history of its path from source to the current node is maintained and saved within the message. Thus, when a request reaches the destination, it will contain the path followed up to that point. This information will be used by its source, when the reply will arrive to it, as well as the intermediary nodes, in order to determine the return unicast forwards.

D. Routing

Whenever a node N from a zone Z needs to forward a request message towards its destination, N will check his zone routing table (next[destination]) and will forward the message only to his neighbours that are assigned to the zones specified in that table (including Z).

Routing is aborted if the current node is already in the path (and adding it again would determine a loop) of if its TTL reaches 0.

After reaching the destination, each message is returned as reply (with a complete path between source and destination). In this phase, all the forwards are unicast, because each intermediary node is able to identify the next hop from the path field of the message (or by using the last[source] information, stored locally).



Status: Connection established with everyone. You may start writing.

Find routes to: 5 TTL: 4

```
Flights between 0 and 6:

Path 1. 0 - 3 - 4 - 5 - 6 -
Path 2. 0 - 1 - 4 - 5 - 6 -
Path 3. 0 - 1 - 2 - 5 - 6 -

Flights between 0 and 5:

Path 1. 0 - 3 - 4 - 5 -
Path 2. 0 - 1 - 4 - 5 -
Path 3. 0 - 1 - 2 - 5 -
```

Fig. 1. Interface with the client

E. *Sample application*

We have developped a simple application that implements this algorithm, but only at a small scale (7 nodes), for correctness checks. Figure 1 shows a screenshot of the user interface with one server. The IDE used was NetBeans. [1]

III. Conclusions

Although quite simple, the proposed algorithm proves to be efficient in terms of bandwidth usage and computational overhead implied, especially when dealing with large scale networks. Future improvements still need to be implemented, such as local caches for recent queries and so on.

References

[1] NetBeans IDE, netbeans.org