

User's manual

User's manual

Table of Contents

1. Introduction	1
2. Start	2
3. Customization of the simulator	4
Customization of the bootstrap mechanism	4
Customization of the stats maker class	4

Chapter 1. Introduction

This document provides an explanation on how to use the peer-to-peer simulator.

Chapter 2. Start

We're going to start from the main method and work back to the inner workings of the simulator. In this way it will be possible to produce examples that help learning to use the simulator.

The main class is called `com.sun.dbe.p2psim.Simulation` and has a `main()` method. This class's main method accepts a series of parameters, many of which have default values to let you run a dummy simulation. From there you can put your own parameters and extension classes until you simulate your own peer-to-peer network.

The list of options is:

- `-h | --hostname`: The hostname the nodes will belong to. All nodes will belong to the same host. The hostname is irrelevant in this release of the simulator. Defaults to `localhost`.
- `-n | --nodes`: The number of nodes to create. It defaults to 1000.
- `-b | --bootstrap`: The bootstrap class. This class is used whenever a new node joins the peer-to-peer network and has to find a node to connect with. It defaults to a class that first selects one of the most connected nodes, following a probabilistic approach where the probability for a node to be selected follows a power law with respect to the node degree. Then it performs a random walk three hops away. The random walk may come back to the same node. In this way it is avoided to always select the same nodes, a condition known as *rich get richer*.
- `-r | --registrations`: The number of registrations to be performed. It defaults to 30.
- `-t | --ttl`: The time to live of each lookup request. Indicates the radius of the search from the initial lookup node. It defaults to 10.
- `-s | --statsmakerclass`: The class that calculates statistics on the graph that results after the simulation. Defaults to `com.sun.dbe.p2psim.stats.StatsMakerImpl`. It calculates the degree distribution of the resulting graph and writes it to the output file in a CSV format.
- `-l | --lookup`: The number of lookup operations to perform in the simulation. It defaults to 1000.
- `-o | --out`: The output file name. It defaults to standard output. If "-" is specified, it will also be standard output.

The simulation goes like this: First, the specified number of nodes is generated. Then, the nodes are added to the peer-to-peer network. For each node added the bootstrap mechanism class is used to select a node on the network to connect to.

When all nodes have been added, items for registration are generated. The number of items generated equals the number of registrations to be performed. The item generation is not configurable, as it should have a minor impact on the simulation. A match is a match no matter what the peer-to-peer network looks like.

Then each item is registered once and only once. To register each item the bootstrap mechanism is used again, to find a node where the item is to be registered.

When all items have been registered, the lookup phase is executed. The lookup step is performed as many times as specified. In each step the bootstrap mechanism is used to select a node where the lookup will be performed.

In each lookup performed on a node the behaviour of the lookup is governed by the node. The request is forwarded to all nodes. This behaviour will be configurable via a user class in a future release of the

simulator.

When the lookup phase has finished, the statsmaker class is called to calculate statistics on the resulting graph. The default implementation calculates the degrees of each node. Then the degrees are normalized to the greatest degree of the network. The resulting data is written to the output file specified in CSV format. This CSV format can then be used with analysing tools, or plotted via gnuplot or similar tools.

Chapter 3. Customization of the simulator

We have already seen that the simulator uses some classes to perform various steps of the simulation. These classes can be substituted by user implementations to modify the behaviour of the simulator.

This behaviour may seem cumbersome for non-programmers. It is. But the alternative was to standardize a set of mechanisms and policies and let the user decide which ones to use. While simpler, it also provides a less powerful method, because there is no way to implement mechanisms or policies fancier than the designer of the simulator could think of.

Another, still less attractive, approach, would have been to provide a programmatic extension different from Java classes. In this approach either a ad-hoc scripting language could have been used, with the drawback that it would not be standardized and had to be learned separately. Choosing a well-known scripting language is a better alternative, but not much different than using a full-fledged programming language such as Java.

Customization of the bootstrap mechanism

The `bootstrap` mechanism class is an implementation of the interface `com.sun.dbe.p2psim.bootstrap.BootstrapMechanism`. The simulator obtains an instance of that class through a class named `com.sun.dbe.p2psim.bootstrap.BootstrapMechanismFactory`. The `--bootstrap` option passed to the main simulator class is the name of the `BootstrapMechanismFactory` subclass. If you want to customize the bootstrap mechanism, you want to start by subclassing both `com.sun.dbe.p2psim.bootstrap.BootstrapMechanismFactory` and `com.sun.dbe.p2psim.bootstrap.BootstrapMechanism`.

The factory has only one abstract method, `getMechanism()`, which returns the instance of `BootstrapMechanism` used in the simulation to find nodes. You want to extend the `BootstrapMechanismFactory` class and provide the name (together with its package name) to the simulation main class, via the parameter `-b` or `--bootstrap`.

The `BootstrapMechanism` interface has only two methods, `join()` and `find()`. The first method is called by the simulation class whenever a new node is added to the graph. In this way you have an opportunity to note the arrival order of the nodes, perform probabilities calculations or whatever you think could be of interest.

The `find()` method returns an arbitrary node that exists in the peer-to-peer network. To make sure that the node exists in the graph the totality of the existing nodes is provided as a parameter to this method. You may choose whatever selection method you want, as this is one of the points of a peer-to-peer network that may be improved.

Customization of the stats maker class

When the simulation finishes, the stats maker class is called. To provide your own implementation of the stats maker class you have to implement the `com.sun.dbe.p2psim.stats.StatsMaker` interface with a class that provides a no argument constructor. In a later implementation of the simulator an abstract factory will be used.

The `StatsMaker` interface only provides one method, `makeStats`. This method receives an instance of `Graph`, and an `OutputStream`. You can do whatever statistics on the graph as you like. The statistics are for the moment limited to the final state of the graph (you can not know how many hops lookup requests jumped on average). If greater capabilities are needed for the simulator to be useful, a later implementa-

tion may provide different mechanisms.