T-110.5150 Applications and Services in Internet

Assignment 1

Peer-to-Peer Network

Contact: t-110.5150@tkk.fi

Deadline: 23.59, Helsinki time, 31, October, 2011

V1.0
Aalto University
Data communication Software Lab
2011

1. Description

In this assignment, you will form a small team (maximum 2 members) to implement a P2P application based on a given protocol. The major functionalities of this application include:

- Join the internal P2P network for this assignment;
- Publish contents in your node and make it visible to other nodes at the same P2P network;
- Look up specified content in the P2P network

In order to test and observe the behaviour of P2P nodes, an internal P2P network is constructed. This network is Gnutella-alike style (and simplified version), without central facility. Each node uses a piece of bootstrap information to join the network. In order to communicate with other nodes, your application should fulfill the requirements of the protocol. For the details of the protocol, please refer to the next section. Grading is based on numbers of features your application provides, the final report and the demo. Please read grading section for more information.

2. Protocol specification & behaviours

The protocol for this assignment is a binary protocol and originated from Guntella 0.6[1]. This protocol is located on top of TCP and contains a 16 bytes fix header. Similar to other binary network protocol, all value fields conform to network byte order. In order to differentiate from TCP packets, we use the term “message” in our protocol.

Node identifier

To identify different nodes, each node needs a global unique ID. In the protocol, we use the combination of node IP and its listening port as node ID.

Protocol header

The following graph illustrates the structure of protocol headers:

```
0                        16                       32
+-----------+-----------+-----------+-----------+
|  Version   |    TTL    | Msg Type   |Reserve bits|
+-----------+-----------+-----------+-----------+
|      Sender Port       |      Payload length    |
+-----------+-----------+-----------+-----------+
|           Original Sender IP Address            |
+-----------+-----------+-----------+-----------+
|                   Message ID                    |
+-----------+-----------+-----------+-----------+
```

Below is the explanatory information for each field in the header:

Version: The version of this protocol, currently it is always one. Any message with a version value other than one should be dropped.

TTL: Time To Live. This Field must be less than or equal to five. Each time when a message is forwarded, this field must be decreased by one. If TTL of one message equals to zero, this message must be dropped.

Msg Type: The types of message, valid values are:

- 0x00 Ping
- 0x01 Pong
- 0x02 Bye
- 0x03 Join
- 0x80 Query

- 0x81 Query Hit

Original Sender IP Address: The IPv4 address of the sender who originally sends this message. The intermediate nodes should not change this field when forwarding the message.

Sender Port: The listening port number of the originally sender.The intermediate nodes should not change this field when forwarding the message.

Message ID: The message ID should be globally unique for each message. One recommendation to generate the ID is put the IP of the sender, port, time stamp and a sequence number together, then hash them together to form a message ID.

Payload length: the length of payload in bytes. The header length is NOT included.

Reserve bits: Set these bits to zero, they are not used in this version.

Types of Message

0x03 Join

The Join message has two sub-types, Join Request and Join Response. Join Request has no body (payload equals to zero). Join Response has a 2 bytes body, which contains the result of Join Request.

A new node sends Join Request to a node already residing in the network for the permission of entering the network. This is the so-called bootstrap process. The connection information (IP and port) of the existing nodes are provided from other sources.

A node already joined the network can also send a Join Request to another node, in order to expand its own routing table. Through connecting with more nodes, the stability is guaranteed.

If the Join Request is accepted by a remote peer, it will return a Join Response message with the same message ID and a 0x0200 status code in the message body. After that, both nodes should consider each other as a normal peer and keep the TCP connection for message exchange.

In version one of our protocol, there is only one valid status code (0x0200). All other codes are invalid and how to handle these codes are purly based on implementation. To refuse a Join Request, a standard way is to close the TCP connection.

Message body of Join Message response: (the length of the body is 2 bytes)
```
0                          16                         32
+------------+------------+------------+------------+
|       Status code        |                         |
+------------+------------+------------+------------+
```
For instance: the message body of a Join acceptance will be:
```
0                          16                         32
+------------+------------+------------+------------+
|     0x02   |     0x00    |                         |
+------------+------------+------------+------------+
```

0x00 Ping

There are two purposes for a Ping message:

A) Availability testing(heart-beat testing)
- A Ping message with TTL equals to 1.
- This message is optional, but recommended to send it every 5 seconds.

- Peers who receive this Ping message must respond with a Pong message (if this peer supports Pong message).
- If no Ping or Pong message is received from the remote side for several times, a node can consider there is some problem with this peer. However, the node should still keep this connection, in case the peer does not support Ping and Pong message.

B) Network probing

- A Ping message with TTL larger than 1.
- This function aims to find more peers based on existing nodes.
- A node (if Pong message is supported) receiving this kind of Ping message should respond a Pong message which contains maximum five entries of its neighbor peers (not include the one who initiates the Ping message). A node may have more than five entries available, how to choose five from all the neighbors depends on implementation. Although a good selection strategy can greatly enhance the robustness of the network. For simplicity, you can choose randomly or just first five.

A Ping message always contains NO message body, the only different between Ping type A and B is the TTL field.

0x01 Pong

Correspondingly, a Pong message also has two types.

- A Pong message for responding availability testing. This Pong message carries NO body information.
- A Pong message for responding network probing. This Pong message contains its neighbor peer information in the body. The body of the message consists of: total number of the entries and one entry record(neighbor IP and listening port) for each neighbor.

Message body of Pong for network probing:

```
0                                  16                                 32
+------------+-----------+------------+------------+
|        Entry size       |           SBZ          |
+------------+-----------+------------+------------+
|               IP Address of the 1st Entry        |
+------------+-----------+------------+------------+
|   Port of 1st Entry     |           SBZ           |
+------------+-----------+------------+------------+
|               IP Address of the 2nd Entry        |
+------------+-----------+------------+------------+
|   Port of 2nd Entry     |           SBZ           |
+------------+-----------+------------+------------+
|                     …  … … … … … …                |
+------------+-----------+------------+------------+
```

- **Entry size:** the number of entries in the Pong message body.
- **SBZ:** Should be zero. These bits are reserved.
- **IP address of Xnd Entry:** The IP address of Xnd Entry.
- **Port of Xnd Entry:** The listening port number of Xnd Entry.

0x80 Query

A Query Message contains query key in the message body. It is propagated in the network according to its TTL (forwarded by peers). A node receiving this message should look up its own sharing data, and return matched entry (if any) in a Query Hit message, and it should also forward this message to its neighbor peers (if TTL larger than 0) no mater the query is hit or not in its local share files.

The length of search criteria varies, and the actual length is decided by the Payload length field in the message header. Nodes should NOT change the message ID of a query message while forwarding.

If a node receives a Query Message whose ID is identical to a previous query in a short time period, the node should drop this message.

Message body of a Query Message:

```
0                        16                       32
+-----------+-----------+-----------+-----------+
|                                               |
|     Search criteria (variable length)         |
|                                               |
+-----------+-----------+-----------+-----------+
```

0x81 Query Hit

Query Hit message uses the same message ID as the Query message. It is routed back to the query sender using the reverse path of the Query message.

The body of Query Hit message contains entries of matched resources, each entry provides information: resource ID (unique for the sharing node), resource value. Like Pong messages, the first 2 bytes of a Query Hit message contains the total number of entries in the message.

Message body of Query Hit:

```
0                        16                       32
+-----------+-----------+-----------+-----------+
|       Entry size      |          SBZ          |
+-----------+-----------+-----------+-----------+
|    Resource ID (1st)  |           SBZ          |
+-----------+-----------+-----------+-----------+
|               Resource Value (1st)            |
+-----------+-----------+-----------+-----------+
|    Resource ID (2nd)  |           SBZ          |
+-----------+-----------+-----------+-----------+
|               Resource Value (2nd)            |
+-----------+-----------+-----------+-----------+
|                  …  … … … … … …               |
+-----------+-----------+-----------+-----------+
```

- **Entry size:** the number of entries in the Query Hit message body.
- **SBZ:** should be zero.
- **Resource ID (Xnd):** The ID of the resource.
- **Resource Value:** The value of the resource.

0x03 Bye

Bye message is Optional with TTL equals to 1. The receiver should not forwards this message and should close the TCP connection immediately. The sender should wait for the termination of the connection and then quit the network. Bye message takes NO message body.

3. Detailed instructions & tips

Network Byte Order

The P2P protocol is a binary protocol, and like other network protocol, it follows network byte order. The network byte order is big-endian. However, the machine you use for programming is normally x86 architecture. In other words, it is little-endian. Remember to handle the endianness during programming (if you use C programming language). More information of endianness can be found at [2].

How to bootstrap?

Bootstrap information will be provided in the course notice board (Noppa, also via Email). The bootstrap information contains existing nodes (IP address and listening port) in the network. A new node can send Join Message (refer to Section: Protocol specification) to this IP and port combination thereby participating in the network.

Should I implement the full protocol specification in this assignment?

Implementation of the whole protocol is not compulsory. Only fulfill parts of them are enough to accomplish the two tasks (refer to Grading section). However, you are encouraged to explore more in your application.

Should I consider NAT or firewall problems?

In this assignment, we assume that there are no NAT boxes or firewalls between any two nodes, so you do not need to consider them. In the Gnutella v0.6 protocol, a special PUSH message is adopted to handle these problems. Read the Gnutella RFC for more information.

Should I provide GUI?

The purpose of this assignment is not fancy GUI, however if you want to try GUI programming, feel free to do it.

Should I use multi-thread/process?

The assignment is carefully designed so that it can be achieved with a single process. The programs for different demo task can be different applications, as long as you can achieve the goal. Of course we recommend you to implement a fully functioning node, but it is not mandatory. A related I/O API you might be interested in is called “select”, which is both supported by C and Java.

Which development environment and programming language can I use?

We highly recommend you to use C under Linux/Unix environment. If you are not so familiar to network programming, learning it from C enables fully grasp of the network programming concepts. Conversely, high level languages, normal coming with highly wrap libraries, hide those things from you.
Another alternative language can be considered is Java. Since this assignment is introduced from this year, we can only provide the protocol specification and some utility functions in C language. If you use Java, you need to implement them by yourself.

How to locate the problem in my code?

For network programing, Wireshark or Tcpdump is a very nice tool for debugging. Observe the inbound and outbound packets to check if you really send or receive the messages and weather they satisfy the format of the protocol.
The common debugging tool for C is GDB, which enables you to set break point in you application and run the program step by step. A light version of starter guide can be found at [3]. There is also similar debugger in java, and can be easily launched in IDE like Eclipse or Netbeans. Sometimes, printing out the information also helps you solve the problem.

Think before coding

Conquer the feature list one by one without having a full view is not recommended. Eventually you may find that you have to rewrite your application in order to implement the next feature. Understand all features and have a clear design can increase your efficiency.

4. Submission

At the end of the assignment, you need to submit:

```
src/
README
Report_1.pdf
```

NOTE:
- The src folder should contain all you source code.
- The README file is a text file and it should explain how to compile and run your application.
- Clearly state your solution for this assignment in Report.pdf. Please be concise and directly go to the main points. The report should be no more than 4 pages.
- Please follow the file structure and naming convention, otherwise your submission will be ignored.
- Submit you source code, NOT binary executable file.
- If you have other files need to be included, please also explain why you need them in the README file.

Each team will be assigned a repository from version control system (SVN), for source code management and submission.

5. Demo

The Demo time is 30 minutes sharp, which includes 5 minutes for setup, 20 minutes for presenting the application and 5 minutes for questions. All members should attend the demo session.
The basic requirements are:
- Show the application is able to search a specified key in the network, and get the value of the specified key.
- Show the application is able to publish a specified key in the network.

You are free to decide how to present other features of your application, but remember to make a plan and finish your demo on time.

6. Grading

The assignment is split into several functions, and points are given separately to each function.
To pass the assignment, you have to implement all features in the basic level. Basic level features also satisfy the basic requirements of demo sessions.

Total: 130 Points
0 - failure (<20 points)
1 - (>=20 points)
2 - (>=40 points)
3 - (>=60 points)
4 - (>=80 points)
5 - (>=100 points)

Basic (20 Points)

Send Join Request message in order to join the network	5 Points
Handle Join Response message, considering the situation when the bootstrap server closes the connection	5 Points
Send Query message with a searching key	5 Points
Publish a key/value pair in the network. This is achieved	5 Points

by responding to a Query message when its search key equals your publish key.	

Advanced (80 Points)

Respond Type A Ping message with Pong. Tell you neighbor that you are still alive.	10 Points
Send Type A Ping message. Check if your neighbors are alive.	10 Points
Replying Type B Ping message with Pong. Tell your peer who else you know.	10 Points
Sending Type B Ping message to find more node information in this network	10 Points
Join multiple peers in order to expand your network	10 Points
Forwarding Query message (consider loop avoidance) to help other node to find a resource	15 Points
Forwarding Query Hit message by following the reverse path of its corresponding Query message	15 Points

Other (30 Points)

A clear final Report State how you solve the whole assignment. Comments on this assignment are also welcomed. No more than 4 pages and don't paste source code.	10 points
A good demo Show good understanding of the assignment, application running well and finishing the demo on time.	10 points
Extra points for constructive comments on this assignment, a good implementation, or other unexpected but decent outcome.	10 points

Contact us: t-110.5150@tkk.fi

Reference

[1] Gnutella Protocol v0.6, http://rfc-gnutella.sourceforge.net/src/rfc-0_6-draft.html
[2] Endianness, http://en.wikipedia.org/wiki/Endianness
[3] Basic GDB Tutorial, http://www.dreamincode.net/forums/topic/13243-gdb-tutorial/