Gustave Eiffel University
Request For Comments: 9384
Category: Informational

Ramaroson Rakotomihamina Johan
Tellier Quentin
February 2023

Greed Protocol -- Greed/1.0

Status of this Memo

This memo provides information about the UGE Greed Protocol. However, it does not specify any details of the implementations or technologies used.

Summary

The UGEGreed Protocol is an application-level protocol intended to create a distributed computing system built on top of the TCP protocol. The main idea is to provide an effective tool for researchers who want to test conjectures on a very large number of cases by distributing their computations over several machines.

Table of contents

1. Purpose

The purpose of this protocol is to distribute a large computation across multiple servers that are linked to each other. Each computation application can be linked to another and then become part of the distributed computation system.

2. Overview

The Greed Protocol is based on the TCP protocol. Therefore, a Greed connection is in fact a TCP connection inside the distributed system and between the client and the application receiver. In that case, when a packet

is sent, the protocol ensures that it will never be lost.

Every strings in the packet of this protocol must be encoded in UTF8 before sending, including the result of a conjecture and the specified address of the client.

In this memo, the notion of parent and children will be mentioned. From an application's perspective, a parent is another application to which it decides to connect. On the other hand, a child is another application that has requested to connect with it.

Every packet in the Greed protocol begins with an OpCode, which is represented by a single byte. An OpCode gives information about the type of packet that was sent. The meaning of each OpCode is detailed in the annex of this document.

A Packet shared inside of the distributed system must contains the address of the client and its conjecture id to identify the source. Please note that each conjecture's identity must be provided by the client as an integer and this identity must be unique across all the conjectures that this client sends.

The Greed protocol ensure that the client will always get all of the requested conjectures unless the client decides to terminate the connection himself or if the target application responds with an op code error, indicating that the application didn't attempt to compute the conjecture.

This protocol also ensure that the system will remain connected even if an application disconnects. The detail of the instructions will be mentioned later in this memo.

Since the performance of the distributed system mostly depends on the number of linked servers, a limit has to be set for each application running in the system, about the number of computation that a server can treat, to avoid overloading of the system.

3. Server

3.1 Setup

A server can be launch in two differents modes:

- Root : the root of the system. There is only once root per distributed system.

- Listening: A server linked to another, it becomes a part of the computing system and extends the networks.

Any server can be linked to another and any server can accept and

distribute conjecture from the client.

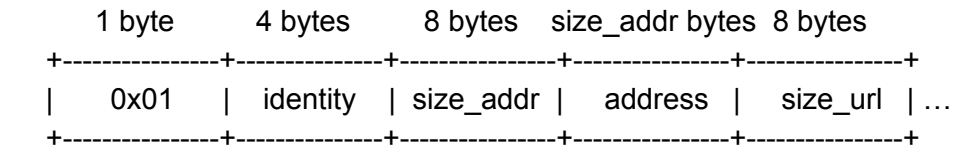
By specifying one port for the current listening port of the server, the application is launched in Root mode. Therefore, if the application is launched giving two ports, then it becomes a listening application. When a server in ROOT mode is shut down, all the servers that are connected to the networks will also be shut down.

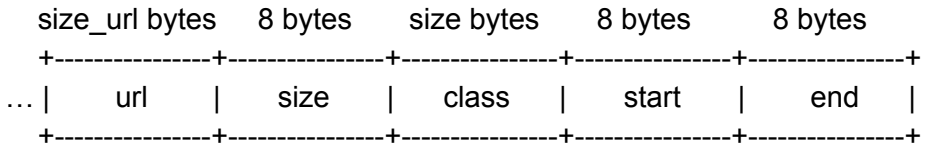
If an application receives two different conjecture from the same IP address but the conjecture ID exists (i.e a conjecture with the specific id for the specific IP address is computing), the connection will be ignored and closed.

3.2 Distribution

The server will distribute the interval of computation he receives, dividing by the number of his children + himself and his parent if he is not the root application.

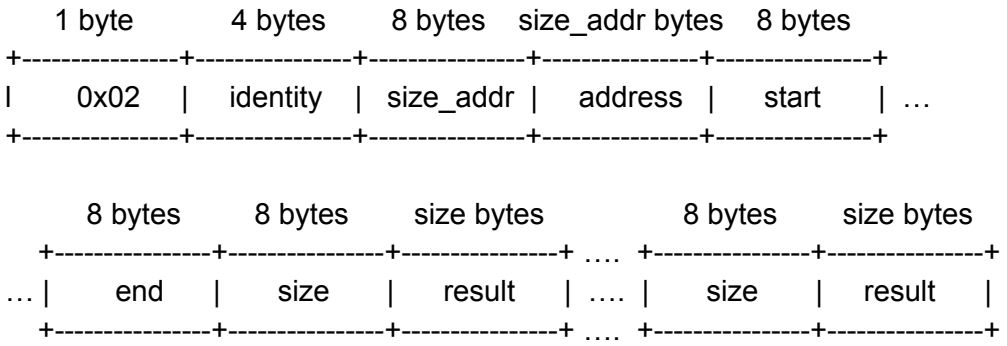
The packet will contain:
- code 1 as a byte
- identity of the calculation as a int
- size of the address as a long
- address of the client
- size of the url of the jar as a long
- url of the jar encoded in UTF-8
- size of the name of the class as a long
- name of the class encoded in UTF-8
- start of the interval as a long
- end of the interval as a long





The server will answer to the application that sent him the packet to compute with a packet that contains the answer of the interval:
- code 2 as a byte
- identity of the calculation as a int
- size of the address of the initial client to identify him as a long
- address of the client

- start of the interval calculated as a long
- end of the interval calculated as a long
- for each end - start strings the size as a long and then the string encoded in UTF-8



When an application gets an answer he will bring up the answers to the server that receives the request from the client and this server will send it to the client.

When a server is overworked he will just distribute the computation to his childrens and will not take any part of the interval, if the server has no childrens he will send it back to his parent until a server is free to do the calculation, a server is overworked when he gets more than 2 000 000 000 values.

3.3 Response to the client

Once an application that received a conjecture received results from the other application, he has to send a packet containing those results and the range of the computation to the client.

The packet of the response will contains:
- code 3 as a byte
- identity as a int
- start of the interval as a long
- end of the interval as a long
- then end - start values that is a size of the string as a long and the string encoded in UTF-8

The description of the packet is in 5.1.1.

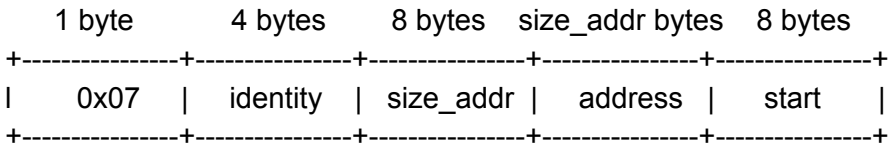
3.4 Clean disconnection case

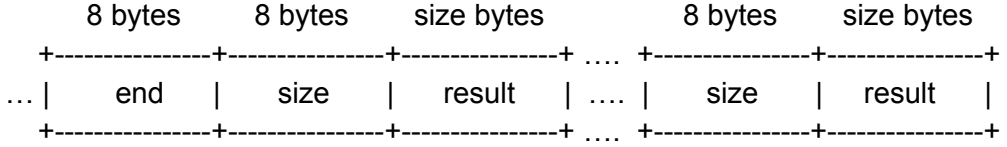
When an application receive a disconnected command, he will perform a series of operations to ensure consistent and stable state of the system.

First, he will update its state from “ACCEPT” mode to “REFUSE” mode, to prevent the packet asking for a computation or a connection and refusing

those by sending an error response with a packet containing only the 0x06 byte. For each conjecture, he must send the following packet containing the interval of the response that it has already computed to each source of the distribution.
Second, this application has to send to each of its child, the address and listening port of its parent, then each of them will perform a link demand with the LINK packet to it.

When a server receives the distribution of a computation but he is shut down, he sends a packet error to the server that asks for the computation, and this server will compute himself.

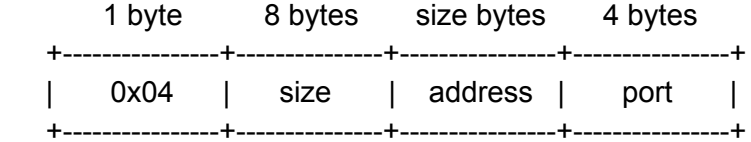




When a server that has to send the response of a computation to a client is disconnected he will also send the packet of the new address to connect to the client, and the client will send the missing interval with the packet in the 5.1.2 section.

The packet to send the address of the parent server for the client and servers:

- code 4 as a byte
- size of the address as a long
- address encoded in UTF-8
- port as a int



When a ROOT server is disconnected he will send a packet to all the servers of the network, so no servers will take new calculations from a client, the servers will end their calculation and then will be shut down.

The packet to send when the ROOT is shut down only contains the 0x05 byte.

```
+----------------+
|      0x05      |
+----------------+
```

When a server can't take a new calculation from clients but receives one he will send a packet with an error code so the client knows he has to send his calculation to another server.

```
      1 byte
+----------------+
|      0x06      |
+----------------+
```

4. Client

First of all, the client has to set up its function which will be computed by the server into a packaged jar which will contains a single class implemented by a functional interface which takes an int as a parameter and then returns a String. The client needs to connect to a server that is part of a computing distributed system. To do so, he needs the following information such as the address and port of the target server. He also needs the name of the final generated file containing the results of the conjecture.

Then he has to send a packet containing information about the conjecture to compute :

- code 0 as a byte
- identity as a int
- size of the url of the jar as a long
- url of the jar encoded in UTF-8
- size of the name of the class as a long
- name of the class encoded in UTF-8
- start of the interval as a long
- end of the interval as a long

The content of the packet is detailed in the 5.1.2 section.

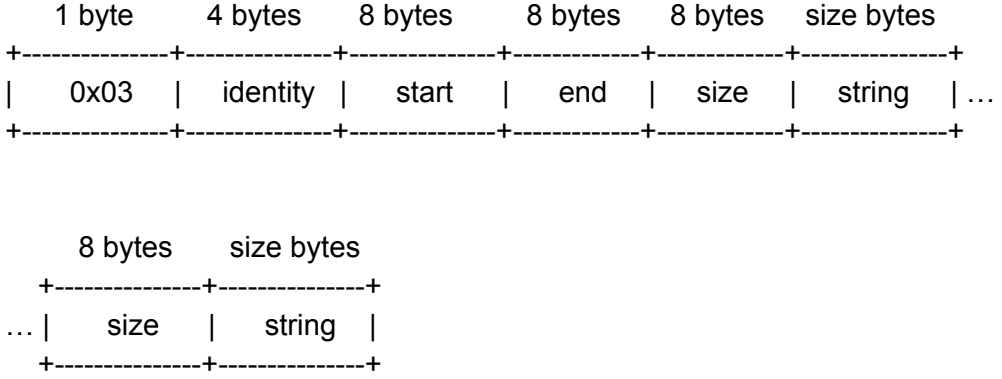
Once the conjecture is sent, the client has to wait until he receives responses. The expected packet is the 'Conjecture response packet' precisely described at the 5.1.1 section. Then, the client has several information about the response, such as, the range of the treated, and each line of the computation. This protocole ensure that the computation of a conjecture will be over and complete no matter what happens inside the system. Therefore, the client can wait until every range he receives

corresponds to the full demand.
Once he finally received all the conjecture, he generate a file containing all the received results are named by the given name in the beginning or by default ‘result.txt’.

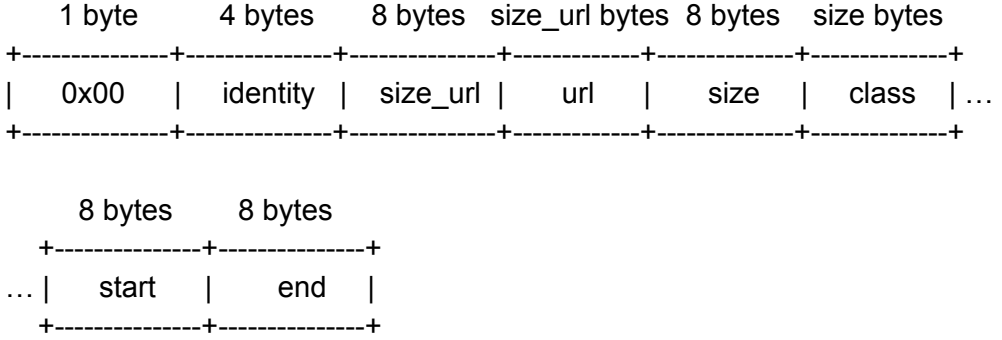
5. Annex

5.1 Packet shared between application and client

5.1.1 CONJECTURE packet



5.1.2 LINK packet



5.2. Op codes

There is different opcodes in the protocol:

- 0x00: is the initial packet that the client send to the server to initiate a computation
- 0x01 is the packet that servers will use to distribute the computation in intervals

- 0x02 is the packet that a server will use to send the answer of an interval to another server
- 0x03 is the packet that a server will use to send the answer of an interval to a client
- 0x04 is the packet to specify a new address to the client when a server is shut down
- 0x05 is the packet that the ROOT server send to all the servers when he is shut down
- 0x06 is the packet that a server send to a client or a server when he receive a new calculation that he can’t take
- 0x07 is the packet that a server send when he is disconnected that contains the response that he already computed