Université Gustave Eiffel Yohann REGUEME
Request for Comments: ???? Mathis MENAA
Category: Informational February 2023

UGEGreed Protocol -- V.1.0

Abstract

This document specifies the UGEGreed project, which aims to develop a distributed computing system on top of the TCP protocol.
The purpose of the system is to assist researchers in testing conjectures on a large scale, by distributing computation across multiple machines. The document outlines the key design principles and features of the system, along with any relevant specifications and requirements for its implementation.

Table of contents

1. Introduction

Scientific research often involves complex computations that can require significant amounts of time and resources. In order to improve the efficiency and speed of these computations, the UGEGreed protocol was developed to allow for the distribution of computation tasks among a network of machines. This protocol aims to provide researchers with a more efficient and cost-effective solution for performing large-scale computations.

1.1 Problem Statement

Performing a large number of computations can lead to significant delays for researchers. While modern computers can perform millions or even billions of computations per second, complex computations can still take considerable amounts of time. This can result in delays in obtaining results and hinder research progress.

1.2 Basic Rules

The types defined below, as well as their memory size, will be effective throughout the rest of this documentation.

BYTE = any 8-bit sequence of data.

INT = any signed integer encoded in 4 bytes in BIG-ENDIAN format.

LONG = any signed integer encoded in 8 bytes in BIG-ENDIAN format.

STRING = any combination of variable-length BYTES that represent a sequence of characters encoded in UTF-8.

1.3 Architecture

To use this protocol, the client must start a first application (ROOT). Once started, the client can connect as many applications as desired to the ROOT or to other applications connected to it.

An example architecture is provided below :

```
                         ---------------
                         | Application |
                         |      1      |
                         ---------------

                              | ^
                              | |
                              v |

     ---------------          ---------------
     | Application |  ----------->  | Application |
     |      2      |  <-----------  |    ROOT     |
     ---------------          ---------------

                              | ^
```

```
                                  | |
                                  v |

     ---------------          ---------------
     | Application |  ----------->  | Application |
     |      4      | <-----------   |      3      |
     ---------------          ---------------

                                  | ^
                                  | |
                                  v |

                            ---------------
                            | Application |
                            |      5      |
                            ---------------
```

1.4 Best Practices

When implementing the protocol, it is recommended to avoid initializing applications in a linear chain. Instead, an architecture in the form of a star should be used to optimize protocol performance. This approach can help to distribute workload evenly across the network and reduce the risk of bottlenecks or single points of failure.

Additionally, it can help to improve fault tolerance and scalability, as new applications can be added without significantly impacting network performance.

When designing the star architecture, it will be important to consider factors such as network topology, application dependencies, and communication requirements. Furthermore, it is important to ensure that the star is properly balanced, and that applications are configured to handle varying workload levels. By following these best practices, developers can optimize the performance, reliability, and scalability of the protocol, which can help to ensure its long-term success.

2. Application

To use this protocol, it is recommended to use the characteristics described in this section to build an application that uses this protocol. It is, of course, possible to use a different implementation for the applications.

This protocol was designed with a focus on managing applications and tasks through the ROOT application. Therefore, the packages are intended for this use in the rest of this document.

2.1 general principle

The application expects on its input a listening port on which the application will accept connections from other applications. Additionally, we can provide the address and port of another application.

When an application is connected, we can then send it an URL leading to the download of the conjecture, the name of the program, and a range of values to test for this conjecture. The applications then evenly distribute the tasks and return the results to the application that sent the conjecture so that it can store them in a file. All of this is done completely transparently to the user.

2.2 Security

It appears that your network uses an authentication mechanism based on a password defined by the network initiator. This type of security is commonly used to prevent unauthorized access to network applications and sensitive data. The goal of this mechanism is to ensure that only authorized applications and users can access network resources.

To achieve this, each application must provide a valid password that matches the one defined by the network initiator. If the password is incorrect or missing, the application will be blocked and unable to connect. While this type of security can help protect your network, it is important to note that it can be circumvented by experienced hackers. Therefore, it is advisable to use other security measures such as data encryption and firewalls to enhance the security of your network.

2.3 Options

When starting an application, it is necessary to specify certain parameters. A password is required for connection to the network, as explained in the previous section. A number of calculations to be sent per buffer, which corresponds to the number of calculations to be performed before sending them in groups to the target application, is also needed.

Here is an example of launching an application in a terminal:

```
path/to/application 6666 192.168.1.255:7777 "topsecretpswd" 3
```

3. Packets overview and functionality

3.1 Connecting an application

When a new application is instantiated in the network, it must make itself known to the network's ROOT. To do this, the new application sends a frame describing its path through the network as well as its address. The ROOT then sends a packet that contains:

- The OPCODE 0, which corresponds to a new connection.

- A number indicating the number of bytes to store the password.

- A password to connect to the network.

- A number indicating the number of bytes to store the address and the port of the new application.

- The address of the application followed by the port in the form "<IP>:<port>".

Next, each encountered application will add a number indicating the size of the next token, as well as its IP address followed by the port in the form "<IP>:<port>".

A packet that goes through 2 applications before reaching the root will have the following format :

```
+------+-----+--------+-----+--------+-----+--------+-----+--------+
| BYTE | INT | STRING | INT | STRING | INT | STRING | INT | STRING |
+------+-----+--------+-----+--------+-----+--------+-----+--------+
```

3.2 Initialize calculation

When an application exists, the client can request it to test a conjecture by providing a URL that leads to the download of the conjecture, the name of the program, and a range of values to test for this conjecture. The application will then create a packet that contains :

- The OPCODE 1, which identifies this packet as an initialization of a calculation.

- A calculation ID chosen by the user, which will allow the result to be returned to the correct application that can write it to the correct file.

- Two numbers representing the beginning and end of the range of values to be tested.

- A number indicating the number of bytes to store the address and the port of the targeted application.

- The address of the application that received the request in the form "<IP>:<port>".

- The download URL.

The size types and element types are detailed in the scheme below :

```
+------+-----+------+------+-----+--------+--------+
| BYTE | INT | LONG | LONG | INT | STRING | STRING |
+------+-----+------+------+-----+--------+--------+
```

This packet will then be sent back to the ROOT to analyze the data and decide whether or not to take charge of the calculation.

3.3 Initialize calculation acknowledgment

When a calculation request is made, the ROOT application sends a response to the application that requested the conjecture to be tested.

This frame contains:

- The OPCODE 2, which identifies this packet as an acknowledgment of the conjecture testing request.

- A number indicating whether the calculation is accepted or refused (with a reason according to the value).

The possible values are:

0: Accept
1: Refuse because the ID is already taken
2: The URL does not allow the download of the conjecture
3: The network is overloaded
4: Others

The size types and element types are detailed in the scheme below :

```
+------+------+
| BYTE | BYTE |
+------+------+
```

3.4 Distribution of calculation

The root will be responsible for distributing the calculations to be performed by the various applications on the network. It will therefore send the following packet to the applications by distributing the tasks so that they can start computing :

- The OPCODE 3 that corresponds to an order to start a computation.

- Two numbers that respectively represent the start and end of the value range to be tested.

- The calculation ID chosen by the user which will allow the result to be returned to the correct application that can write it to the correct file.

- A number that indicates the number of bytes to store the URL.

   - The download URL.

   This is followed by two values that are repeated as many times as
   necessary at the end of the buffer:

   - A number that indicates the number of bytes to store the address
   and the port of an application in the path.

   - The next application address on the path to the target application
   followed by the port in the form "<IP>:<port>".

   This is repeated until the path is complete. An application will
know that the packet is intended for it when no address is mentioned
in the packet.

The packet will have the following format if we need to go through a
single application before reaching the target application :

```
   +------+------+------+---+---+--------+---+--------+-----+--------+
   | BYTE | LONG | LONG |INT|INT| STRING |INT| STRING | INT | STRING |
   +------+------+------+---+---+--------+---+--------+-----+--------+
```

3.5 Send result to ROOT

Once an application has finished its calculation, depending on the
user's choice, the application will send <n> results back to ROOT so
that it can manage the distribution of tasks. The packet will
contain:

- The OPCODE 4 that corresponds to an result.

- The calculation ID chosen by the user which will allow the result
   to be returned to the correct application that can write it to the
   correct file.

   We will then repeat the next two tokens for the number of results
   per buffer requested by the client :

   - A number that indicates the number of bytes to store the result.

   - A result.

   The packet will have the following format if the user choosed 3
   results per buffer :

```
      +------+-----+-----+--------+-----+--------+-----+--------+
      | BYTE | INT | INT | STRING | INT | STRING | INT | STRING |
      +------+-----+-----+--------+-----+--------+-----+--------+
```

3.6 Send result to target application

The ROOT application will then redirect the packet to the target application using the calculation ID. The targeted application will only need to write the result to the output file.

If we go through a single application before reaching the target application, we will have a frame that corresponds to:

- OPCODE 5 corresponds to sending a result to a target application.

- The number of bytes to skip to start reading the IP addresses.

The following two values are repeated for each result:

- A number that indicates the number of bytes to store the result.

  - A result.

The following two values are repeated for each step in the path to the target application:

- A number that indicates the number of bytes to store an adress and the port of the next application.

  - The IP address and port of the next application in the path, in the form "<IP>:<port>".

  The packet will have the following format if the user choosed 1 results per buffer and we need to go through a single application before reaching the target application :

```
+------+-----+-----+--------+-----+--------+-----+--------+
| BYTE | INT | INT | STRING | INT | STRING | INT | STRING |
+------+-----+-----+--------+-----+--------+-----+--------+
```

3.7 Deconexion

It is possible to intentionally disconnect an application that is not the ROOT. Upon disconnection, the application will send the following packet to the ROOT:

  - OPCODE 6 which corresponds to a request for the ROOT to acknowledge the disconnection.

  - The IP address and port of the application which disconnect, in the form "<IP>:<port>".

  The packet will have the following format :

```
+------+--------+
| BYTE | STRING |
+------+--------+
```

In addition, a frame must be sent to the child applications if they

exist so that they can reconnect to the parent application of the
one that is disconnecting. The packet will contain:

- OPCODE 7 which allows to request an application to connect to
another.

- The IP address of the application to connect to in the format
"<IP>:<port>".

```
+------+--------+
| BYTE | STRING |
+------+--------+
```

4. Proposed Concepts

- To improve network performance, one idea is to provide each
application on the network with an indication of the overall network load.
This way, the application can directly refuse to process a calculation it
is offered without having to send the packet up to the root to verify if
the calculation can be handled. By giving each application the ability to
make this determination locally, the network can reduce the number of unnecessary
packets transmitted, which can improve overall network efficiency.
One possible way to transmit this information is through an existing packet
that already circulates frequently on the network, so that applications can
receive the information in near real-time. Alternatively, a dedicated packet
could be created specifically for this purpose, which would be sent to all applications
on the network at regular intervals. Both of these methods would require careful
consideration to ensure that the load indicators are accurate, that the packets do not
cause additional network congestion, and that the information is
delivered in a timely manner.

- To detect when an application on the network has crashed, a reference timeout
is typically used to determine how long the network should wait for a response from
the application before considering it to be dead. However, to achieve more precise
and responsive crash detection, the idea is to refine this timeout dynamically as calculations
are processed. By doing so, the network can adjust the timeout based on the actual workload
of the application and adapt to changes in performance over time. For example,
if an application is processing a particularly large or complex calculation,
the network could increase the timeout to accommodate the increased workload.
Conversely, if an application is idle or not responding, the network could decrease the
timeout to detect a crash more quickly. This approach could improve network reliability
and reduce the risk of lost data or network downtime due to crashed applications. However,
it will be important to carefully monitor and fine-tune the timeout settings to ensure that
they are accurate and responsive to changes in application performance.