CS 6390.001 Advanced Computer Networks

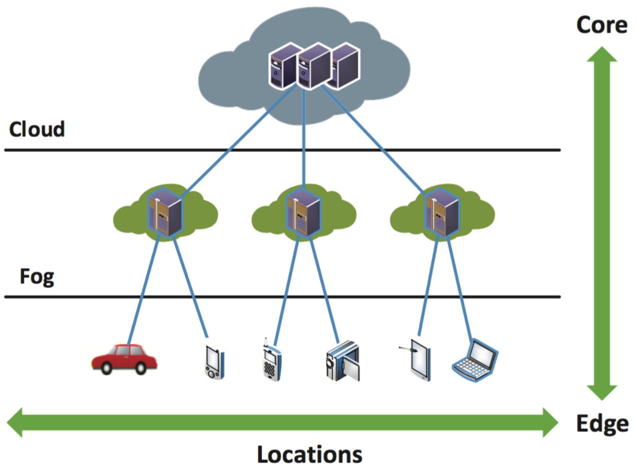
Fall 2016 - Programming Project

Description

Introduction

In this project you will implement functionality of a node in a distributed system in the context of *Fog Computing*. Fog computing is a newly-introduced concept that aims to put *the Cloud* closer to the user for better quality of service (QoS). Fog computing is studied mostly in the context of Internet of Things (IoT) as it provides low latency, location awareness, support for wide-spread geographical distribution, and mobility support for IoT applications. Fog is not a substitute for but a powerful complement to the cloud.

A very simplistic model of a fog is shown in the figure below. In this architecture, the cloud is in the core, IoT nodes are at the edge of the Internet, and the fog sits between the cloud and the IoT nodes closer to the IoT nodes (normally within 1 or 2 hop distance). Fog nodes collaborate among each other and provide relevant cloud services such as computing, networking and storage to IoT devices. In this project, the main use of fog computing is to reduce the response



time delay for requests coming from the IoT nodes.

When a fog node receives a request, depending on its current work load, it will either handle the request by itself or will forward it on. A request can be forwarded among the fog nodes at most *Forward_Limit* times. If a request is forwarded *Forward_Limit* many times and it cannot be served by the last fog node (due to its current work load), the node will forward the request to the cloud for processing. The node processing the request (either a fog node or the cloud) will directly send its response back to the IoT node.

Protocol

According to the figure above, the system includes three types of nodes as (1) IoT nodes, (2) fog nodes, and (3) cloud node. In this project, we will provide you with an IoT node implementation that will be generating requests and sending them to the fog nodes. Your main task will be to implement the functionality of a fog node. You will also simulate the functionality of the cloud node as we will discuss below.

The operation of a fog node includes two tasks: (1) receiving requests and, depending on its response time for this request, processing or forwarding the request and (2) periodically exchanging information with its 1-hop neighbors about their current queueing time so that they know who to forward a request when needed.

When a fog node decides to process a request, it will place it to the end of the processing queue that it keeps pending requests in. The decision on processing or forwarding a request will be based on the expected *response time* for the request. The response time includes queuing delay and processing delay for the request. Each request will include a parameter that will define the processing time of the request. When a new request R_{new} arrives at a fog node A, using the processing time info in R_{new}, the fog node A can calculate the queuing time for R_{new}. The queuing time will be given by the summation of the processing times of all the currently queued requests at A. As a result, the expected response time of R_{new} can be calculated by adding up the queuing delay and the processing time of R_{new}.

When a request is up for processing at a fog node, the processing operation should be simulated by having a thread sleep or wait for processing time of the request. Once this time elapses, a response should be created and sent back to the request originator IoT node.

If a fog node decides that it is too busy to accept this new request (which will be defined by an input parameter of *Max_Response_Time* at each node), it will check if it can forward the request to a neighbor or not. This depends on the *Forward_Limit* and how many times this request has been forwarded so far. If the request has not reached the *Forward_Limit*, the fog node will identify the best neighbor among its 1-hop neighbors and forward the request to that neighbor. The best neighbor is the one that reported the smallest response time in the most recent round with an exception that the best neighbor cannot be the neighbor who has forwarded this request to the current fog node.

If a fog node cannot accept a new request and the *Forward_Limit* is reached for the request, the fog node will need to forward this request to the cloud. In this project, instead of forwarding the request to the cloud, you will simulate this operation in the following way: your fog node implementation will include a thread with a long enough queue which will represent the cloud node. When a fog node needs to forward a request to the cloud, it will put the request in the cloud queue which will incur the necessary processing delay (by using a sleep or wait system call for the duration of processing time recorded in the request message) and then will send the response directly to the request originator IoT node. Note that cloud is assumed to be 100 times faster than a fog node, which means requests will be processed 100 times faster on the cloud. Essentially the cloud is very similar to fog node (a thread with a queue), but it is much simpler than the fog node in implementation. The cloud node processes the requests from the queue one by one in a loop.

Selection of the best neighbor depends on the queueing time of the 1-hop neighbors. This information needs to be periodically exchanged among 1-hop neighbors so that they can use this up to date information to decide on their best neighbors to forward a request when needed. When the time comes to send a periodic queueing time information to 1-hop neighbors (every t seconds), a fog node will announce the queueing time that will be incurred for an incoming request at this very moment and will send it to its neighbors. Once each neighbor does this, everyone will be able to use this information to figure out their best neighbors.

In this project, we will use requests of 10 different types each with a different processing time. A request of type *x* incurs *x* processing time units (assumed to be in seconds). Request type will be decided by the IoT node request originator and will be included in the request message indicating the processing time needed for the request.

Given that a request may visit multiple nodes in the system, for debugging and testing purposes, you are required to come up with a scheme to carry necessary information on the message body. When the corresponding response arrives at the IoT node, the IoT node will print it out. The information should clearly indicate which fog nodes are visited and what each fog node did on the request. This information can be stored as a text string in the message body while it propagates in the system. Each fog node can append its own message to the end of the string. The node sending a response for this request, can copy over the text string into the response message and append its own information to it before sending it back to the requesting IoT node.

To avoid the loop scenario where a fog node sends back a request that it has received from its neighbors to the same neighbor (loop of two fog nodes), you will assume that the fog node offloads the request to its best neighbor, not considering to the neighbor it has received the request from. In this case, if the best neighbor is the neighbor that the request is just received from and an offload is needed, the fog node sends the request to the second best neighbor.

Implementation Details

For implementation, you are allowed to use C, C++, Java, or python. TCP/IP sockets must be used for communication between nodes. All numerical parameters and constants (e.g. *Forward_Limit*, *t*, etc.) of the program must be defined as variables (no value should be hard coded).

We believe that the project involves implementing three protocol operations between the nodes as below:

1) Periodic response time update messages among the fog nodes
2) IoT node to fog node request/response communication

3) Fog to fog request offloading

The operations (1) and (3) above should be implemented over TCP sockets and the operation (2) should be implemented using UDP sockets.

The IoT node request generator will send periodically packets to all of the fog nodes it has received as the input randomly. The IoT node request generator will be run in the terminal with the following command.

```
IoTreqGen interval IP1 UDP1 IP2 UDP2 …
```

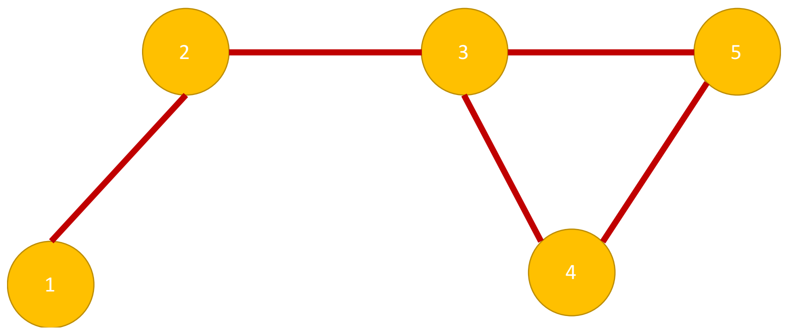
`interval` is the time between sending packets (in ms), `IP1` is the IP address (or hostname) of the first fog node, `UDP1` is the UDP port of the first fog node on which the fog node listens for incoming data from IoT layer. `IP2` is the IP address (or hostname) of the second fog node, `UDP2` is the UDP port of the second fog node on which fog node listens for incoming data from IoT layer. There could be more than 2 fog nodes.

The information that IoT node request generator puts in the packet are:

- Packet sequence number
- Request type
- *Forward_Limit*
- IP address of the request generator
- UDP port number of request generator

Note that the last two pieces of information are necessary for sending the reply back to the IoT node from fog nodes; fog nodes should know where they must send the reply to. This information will be recorded in these two pieces of information in the packet. Also note *Forward_Limit* is included in the requests, so that each request can have a different *Forward_Limit*. Each fog then can check this number and decrement it by one when offloads it, and when it reaches zero, then no more forwarding is allowed.

The following graph describes the topology of the fog nodes in the fog layer (different topology may be used during project demo).



A fog node will be run in the terminal with the following command

```
FogNode Max_Response_Time t MY_IP MY_UDP MY_TCP N1 TCP1 N2 TCP2 …
```

The first two arguments are the parameters of the protocol discussed in this document before. `MY_IP` is the IP address (or hostname) of the fog node, `MY_UDP` is the UDP port number on which the fog node receives requests from IoT node layer, and `MY_TCP` is the TCP port on which the fog node communicates with its neighbors. `N1` is the IP address (or hostname) of the fog's first neighbor and `TCP1` is the TCP port number that neighbor `N1` is listening on to communicate with its neighbors. `N2` is the IP address (or hostname) of the fog's first neighbor and `TCP2` is the TCP port number that neighbor `N2` is listening on to communicate with its neighbors.
Note that one fog node could have more than 1 neighbor. In this case if this fog node is the listener side of socket (i.e. server socket) it should listen on the same port for more than 1 incoming TCP request. (Remember the discussion in class regarding which side of the TCP connection should send the TCP connect() and the other part should accept())

Submission

You are required to come up with a design and submit a design document on e-Learning and meet with the TA to go over the design document. The deadline is October 28 at 11:59PM and you submit this online (no hard copy). The meeting time will be announced shortly.
This is a group project and groups up to 3 people are allowed in one team. For submitting the project, make sure you submit all the source codes to e-learning. (One project submission per group). Be sure to include sufficient comments. Include a README file specifying instructions to run/compile the project, platform, compiler, group information, etc.
Along with the code you are required to submit a short report. The report must in include in high level how you implemented the main functionalities, and any interesting challenge you faced and would like to discuss how you solved it. The report also should include a section showing how each person contributed to the project. The report shall be at most 3 pages.
You will be asked during the project demo session to download your program from e-Learning and run it on some distributed machines for testing. We will run the topology for a certain configuration and go through the node's outputs to see your protocol works properly.