CS4383/5383: Computer Networks

Vanderbilt University, Spring 2026; Instructor: Aniruddha Gokhale

Programming Assignment #1

Handed out Thursday 01/22/2026; Due Friday 02/12/2026 (11:59 pm Central in Brightspace, one submission per team of size 3)

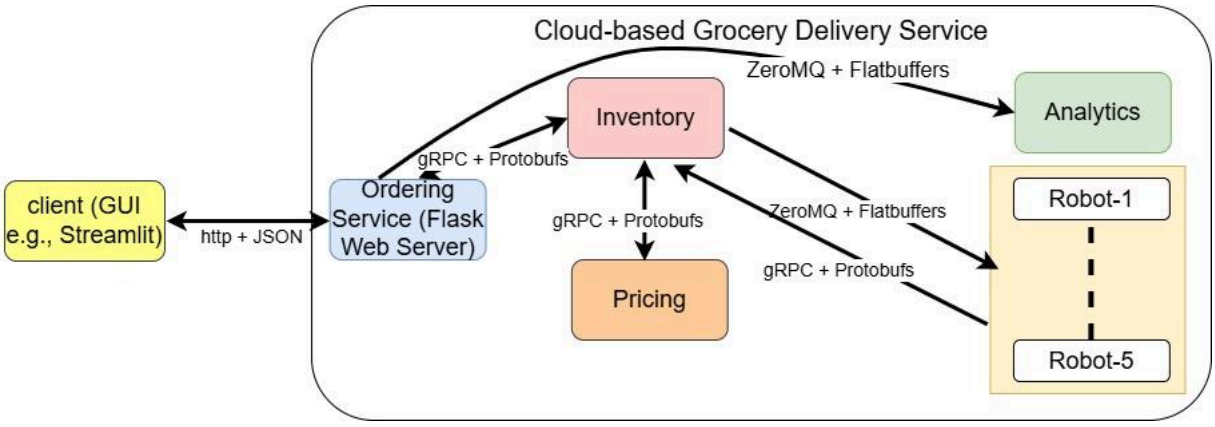
Assignment Theme: Custom Application Protocol with Serialization

Objectives

The goal of this assignment is to develop a microservices-style distributed application that uses a custom Application Protocol (comprising message formats and interactions) that we will define. As part of the overall system design, our aim is to get acquainted with higher-level frameworks to build networked distributed systems, such as gRPC and Zero MQ, in conjunction with serialization frameworks, such as Protocol Buffers and Flatbuffers as well as JSON. We will also get to design a simple web server highlighting HTTP client-server capabilities. Finally, we will familiarize ourselves with deploying these components as simple processes on team-created VMs and single node Kubernetes cluster on laptop VMs (Virtualbox or Multipass).

Scenario: Automated Grocery Ordering and Delivery Service

Our automated grocery ordering and delivery service application allows users and machines (like smart refrigerators) to place grocery orders (which are then assumed to be fulfilled in part or full). Likewise, a supply chain client (e.g., trucks) can restock items within the grocery store ecosystem. The overall architecture is shown below.



Client Category 1: Our first type of client is an Internet of Things (IoT) artifact in the form of an edge-based, intelligent/smart refrigerator. It is edge-based because the refrigerator is installed in homes/RVs, which sit at the edge of the network. It is IoT-enabled because our (hypothetical) refrigerator is assumed to include a variety of sensors and has networking support. Our smart refrigerator periodically sends a

Grocery Order request to the Grocery Service and waits for confirmation. The overall architecture is shown in the figure below:

Client Category 2: Our second type of client is an artifact of the supply chain, where we assume that trucks periodically bring new items to replenish and restock the inventory. We will assume that as a truck unloads the items in the warehouse, it sends a request to the inventory service with details of all the items it got on this trip. Accordingly, the inventory service updates its database and publishes the details to the robots so that they can pick these items from the loading docks and fill up the shelves in their respective aisles.

Service: The Grocery Service itself is internally made up of the following chained microservices:

- A front-end Ordering Service that receives HTTP requests from clients with the grocery order
- A back-end Inventory Service that checks its inventory for availability of the ordered items and subsequently makes request to back-end robots to go fetch the items
- A back-end Pricing service that returns the total price for the items that are still available and will be delivered to the customer
- 5 different robots each responsible for a specific aisle in the grocery store warehouse to fetch (or replenish) the ordered item and inform the Inventory Service that the items are loaded onto the delivery cart (or restocked). The five aisles in our grocery warehouse include aisles for bread (all kinds of bread, bagels, waffles, etc), dairy (different kinds), meat (different kinds), produce (vegetables and fruits) and party supply (e.g., soda, paper plates, napkins, etc)
- A back-end Analytics service that simply keeps track of total number of client requests received, their fulfillment outcome, and the time it took to serve each incoming order (be it a grocery order or restocking order).

Interactions and System Operation

For PA1 and rest of the assignments, when manually testing the working of the system end-to-end, we will be using a browser to manually input either the Grocery Order (sent by smart refrigerator) or the Restocking Order (sent by truck). To test larger bursts of workloads with different arrival patterns, we will create automated techniques such as via *Locust.io*. But we will do that from PA2 onward.

Client 1 (Refrigerator) to Grocery Service: The smart refrigerator periodically sends a grocery order request from a web browser to the Ordering microservice front-end, which serves as the front-end web server. The browser offers a GUI where the user fills out the grocery order and ultimately clicks the send button. Note, the user need not order all items from all five aisles every time; it could be a subset. Note that this will be a HTTP interaction with the grocery order and reply status encoded in JSON.

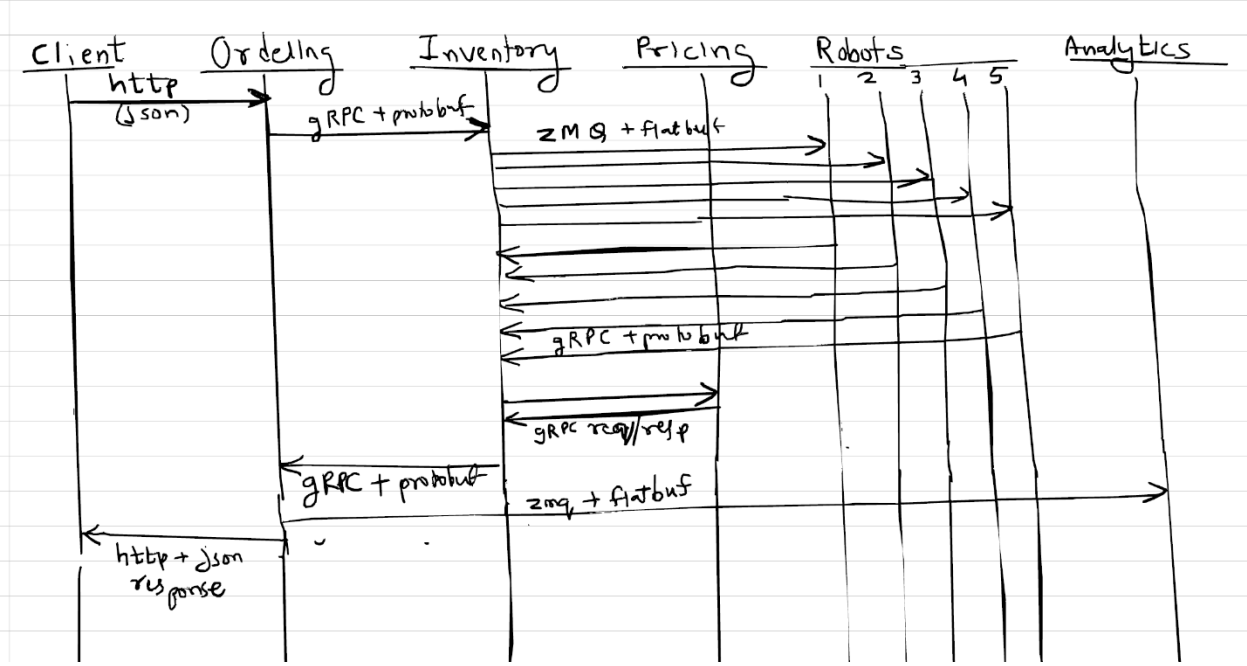
Upon receipt of the message at the Ordering web server, the web server constructs a Protobuf serialized packet out of the received information and makes a gRPC request on the Inventory microservice. The Inventory checks for availability of the requested items in its database and for all those items (and their quantity available), then creates a Flatbuffers serialized message and publishes the “Fetch” message to all the Robot microservices.

The Robot microservices subscribe to the “Fetch” message topic. When they receive the topic, they parse the incoming message and check whether there is an item to be fetched from their aisle. If there is one, then they spend some time (can be implemented using a short sleep () call one for each unique

item to be retrieved. When all items that robot is responsible for are fetched, it simulates time spent in heading to the delivery cart to dump the items into. If a robot has no item to work on, it will not spend any time, and send a simple response to the Inventory indicating a no operation. On the other hand, if a robot actually spends time, after it is done, it will send a success message to the Inventory all using a gRPC/protobuf message. Note that the robot might receive fetch messages for different customers and so when responding, it must also indicate which customer it was serving.

The Inventory service waits to hear from all five robots for each customer request. When all five replies are received, the Inventory microservice will ask the Pricing service to respond with a bill for all the available and ordered items. Once that is received, the Inventory service replies to the Ordering service, when in turn replies to the original client displaying the outcome along with the final bill.

See the interaction diagram sketched as a UML sequence diagram below.



Client 2 (Truck) to Grocery Store

The interactions are very similar. The truck sends RESTOCK message with a supplied ID and the contents to the front-end (although it is called Ordering, it does additional work). Ordering will send gRPC message to Inventory, which in turn publishes a Restock message via ZeroMQ+Flatbuffers to the five robots. The robots will look at the Restock message and check if they are responsible for restocking. If so, they will simulate some time to go to the unloaded items from the truck and put them in the shelves in their aisle. When done, they will send a Success or No-op message to the Inventory. When the Inventory has received messages from all robots for that supplier, it replies to the Ordering microservices and back to the supplier who made the restocking request.

Measuring end to end latencies

For every customer or supplied made request, the Ordering microservice starts a timer and after it receives a response from Inventory, it measures the time. It then constructs an Analytics message and publishes it, which the Analytics component will receive.

Message Types and Reply Codes

Our application will support two types of message types: GROCERY_ORDER and RESTOCK_ORDER (just like GET, PUT etc in http). In Protobuf and Flatbuffers, these two can be defined as Enumerated types. Reply codes consist of OK (indicating that the request is valid and that a valid response is attached to the message) or BAD_REQUEST (indicating that the server received a message that it cannot understand or handle). Even these will be enumerated types.

Student teams are required to define the GROCERY_ORDER and RESTOCK_ORDER message formats for gRPC request-response communication and their responses. Teams must also define the Flatbuffer schema for Inventory-Robot and Inventory-Analytics pub-sub communication, and gRPC schema for Inventory-Pricing, and Robot-to-Inventory communications.

Grocery order must comprise a customer ID, and then the ability to order items across all five categories (bread, meat, produce, dairy and party items). Each category will have sub categories, e.g., produce category can have specific items like tomatoes, onions, apples, oranges, etc and their quantities (say in units of weight). Restock order must comprise a supplied ID, and like the Grocery order, all the categories that the truck is unloading. Note that there must be at least one item being ordered or restocked; the message cannot be empty without any order or restocking.

Define the format for the schema that the Inventory uses to send a publication to all robots and the response sent by robots to the Inventory after they are done with their work.

Technologies used:

- Ubuntu Linux 24.04 LTS (in Multipass or Virtualbox or Chameleon VMs)
- Virtualbox/Multipass instance on laptop or VMs on Chameleon
- Python3
- ZeroMQ, gRPC as communication middleware
- Flatbuffers and Protobuf as serialization frameworks; JSON as a third alternative.
- Flask web server
- Streamlit client-side GUI

Scaffolding Code

First and foremost, please look at gRPC and ZeroMQ tcp_client/server examples in the github scaffolding code. Then, also look at the Serialization scaffolding code for Flatbuffers, ProtoBuf and JSON, and how this can be combined with gRPC/ZMQ send/recv calls. For ZeroMQ, the PUB-SUB socket pairs will be needed for this assignment.

Please see:

- ScaffoldingCode/Serialization at https://github.com/asgokhale/ComputerNetworksCourse.git for the serialization scaffolding. Please note, this is from some years back.

- FlatBuf_ZMQ at https://github.com/asgokhale/DistributedSystemsCourse.git for a full example on combining Flatbuffers with ZMQ Pub/Sub

Allowed Use of LLM/Generative AI

We now live in the age of Generative AI. This technology is now being increasingly used in assisting us in addressing mundane, low-level tasks. Thus, students can feel free to use their favorite LLM for the following tasks:

1. Generating Streamlit code for ordering groceries and restocking
2. Generating Protobuf and Flatbuffer schemas from the schema template you provide
3. Generating Docker and Kubernetes YAML files
4. Any assistance you may need to address concurrency/multithreading support as needed in the individual microservices, however, your team must make every effort to write the code for individual miroservice

Milestones (and hence the deliverables and expectations):

So that students do not start the work at the last minute, each assignment will have three milestones. Although intermediate milestones are not graded, we will keep track of whether the teams are making progress or not. The following provides a rough idea of the minimum that must be achieved by the specified milestone.

Please use the Slack Channel for programming assignments for discussions/questions.

Milestone 1:

- Streamlit-based client-side code for both Grocery order placement and Restocking order. This should be accessible from the browser
- Ordering microservice's basic implementation including interaction with Inventory microservice Inventory microservice's very basic implementation where it receives a gRPC request and just replies with success
- JSON and Protobuf schemas
- HTTP-JSON communication between Client and Flask-based Ordering microservice
- gRPC-Protobuf communication between Ordering and Inventory microservices
- Test that all these aspects work
- Submit to Brightspace what all have you accomplished up to this point and what was not accomplished, and difficulties, if any.

Milestone 2:

- Robot microservice implementation
- Flatbuffers schema for pub-sub from Inventory to Robots
- Protocol Buffer schema for response from Robot to Inventory
- ZeroMQ pub-sub communication from Inventory to Robots
- Submit to Brightspace what all have you accomplished up to this point and what was not accomplished, and difficulties, if any.

Milestone 3 (final): Cloud based deployment

- Full implementation
- Collect end-to-end latency data for the experiments you run.
- Plot graphs
- Create video for the TA explaining all steps, show experiments, describe your schemas etc
- Submit the code, plots etc to Brightspace

Rubrics (for grading after the final milestone)

- Correctness (program works): 40%
- Experiments in all scenarios 20%
- Teamwork (who did what) 10%
- Plots of results 10%
- README file explaining how to run the code: 10%
- Zoom-demo/self-explanatory video to TA or grading peer: 10%

Q&A

Please use the #progassign channel in our Slack workspace.