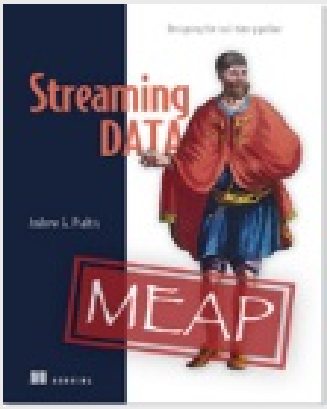

Interaction Patterns: Request/Response

By Andrew G. Psaltis

Andrew G. Psaltis, author of Streaming Data, explains the concept of the Request/Response pattern and how you can collect data using it.

Regardless of the protocol used by a client to send data to our collection tier or in certain cases our collection tier reaching out and pulling in the data, there are a limited number of interaction patterns in use today. Even considering the protocols driving the emergence of the *Internet of Everything* the interaction patterns fall into one of the following categories:

- Request/Response
- Publish/Subscribe
- One-Way
- Request /Acknowledge
- Stream

In this article. We'll take a moment and explore the Request/Response pattern and discuss how we might collect data using it.

Request/Response is the simplest pattern and is used when the client must have an immediate response or wants the service to complete a task without delay. The typical flow for the Request/Response pattern begins with the client establishing a connection to the service, sending a request and waiting for a reply. From the services perspective it processes the request as soon as it has been received and returns a response over the same connection.

Imagine for a moment that we work in the transportation industry and last week while enjoying coffee with our friend Eric, who works in the automotive industry, we came up with an idea to provide a real-time traffic and routing service for all vehicles on the road. Our company would handle building the service and Eric's company would handle building the streaming system that would reside inside the vehicles. We then sketched out what this solution would look like; starting with the vehicle part of it, figure 1 below shows our high level drawing of the vehicle side of things.

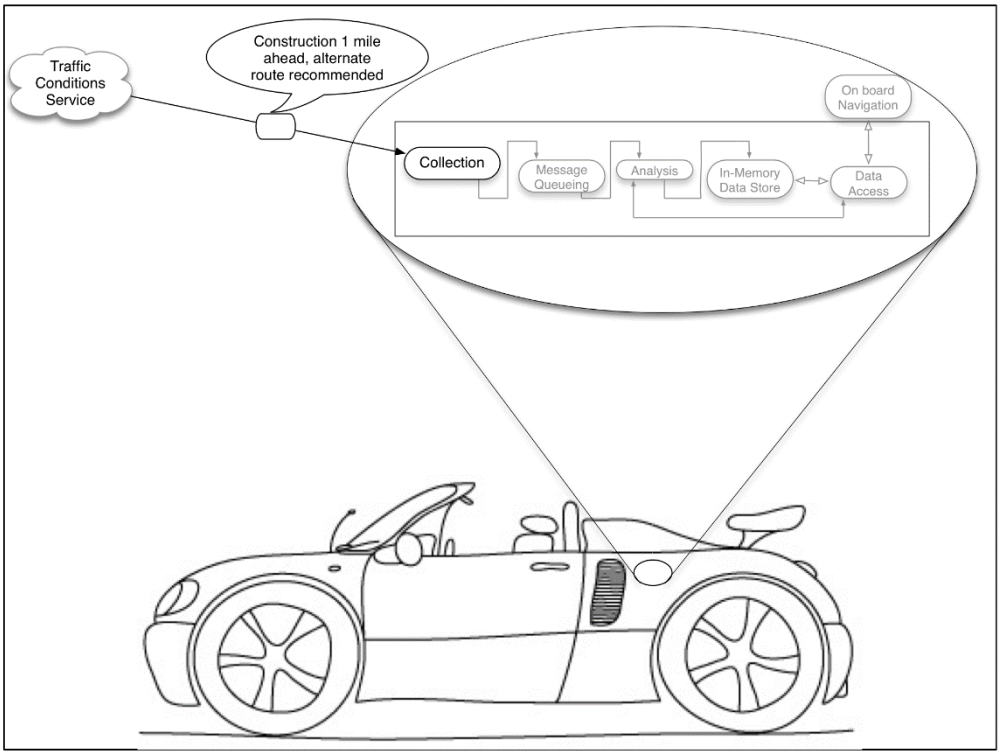

Figure 1 Receiving the response to the traffic conditions request with on-board streaming system.

For the vehicle Eric is going to build an embedded streaming system to not just handle interacting with our traffic and navigation service, but to also have the ability to interact with other services and perhaps vehicles. To accomplish this Eric has decided to use the NVIDIA Jetson Automotive Development Platform (http://www.nvidia.com/object/jetson-automotive-development-platform.html).

Before we dive right in we need to think about the protocol that these vehicles are going to use. At a high level our basic requirements for a protocol are:

- It can be used in an embedded system
- It is lightweight – meaning it does not require a lot of power
- It supports the request/response interaction pattern.

After some research and a dip into the IoT protocol soup, we decided to go with the Constrained Application Protocol (CoAP). This is a RESTful application layer protocol, which naturally supports our interaction pattern. It is designed for very simple electronic devices, it is very lightweight, simple, and runs over UDP. Sounds like it would be perfect for the

embedded car system Eric's team is building and our service. If you want to learn more about this protocol you can find more information here: https://tools.ietf.org/html/rfc7252.

Hands on with CoAP

If you want to get a feel for the CoAP protocol without writing code you are in luck. Matthias Kovatsch, an Internet of Things researcher at ETH Zurich has written a Firefox browser add-on called Copper (Cu). This add-on allows you to interact with CoAP servers from a browser, by simply entering a CoAP URI into the address bar. The add-on can be found here: https://addons.mozilla.org/en-US/firefox/addon/copper-270430/. To install it just go to that URL using Firefox.

With a pretty good idea of the protocol we are going to use we now need to decide what technology we are going to use to build it. Let's briefly outline our requirements for a CoAP library, before we just start searching the Internet for options. As always when looking for third party software there are many options to weigh, in this case if I were building this collection tier my primary requirements would be narrowed down to the following:

1. Written in Java - this is my preferred language to implement my collection tier. Java is not required by any means to build a robust collection tier, I chose it here purely out of personal preference
2. Supports DTLS (Datagram Transport Layer Security) – think SSL/TLS for UDP
3. Closely follows the standards and passes all mandatory and optional protocol tests
4. Is under active development and not someone's side project that has been abandoned

If you search the Internet for CoAP implementations you will find a variety of them. My first choice is to check the CoAP technology website - http://coap.technology/, here we can find more information about the protocol and links to various implementations. Of the implementations mentioned on that site and others, if I were to building this collection tier I would to choose to build it using Californium (https://www.eclipse.org/californium/). This library meets all of my requirements outlined above.

At this point we are ready for Eric's team to go and build the vehicle side of things and we are ready to build the traffic and routing service.

To learn more about this pattern, see Robert Daigneau's *Service Design Patterns* (Addison-Wesley, 2011).