Technical University of Moldova
Faculty of Computers, Informatics and Microelectronics
Department of Computer Science

REPORT

Laboratory no. 4

Processing semi-structured distributed data collections

Author:
Andrei Capastru

Lecturer:
Dumitru Ciorbă

January 21, 2016

Contents

List of Figures

Listings

1 Objective

Use of TCP/IP protocol in the development of distributed applications containing data collections.

— Use UDP protocol in unicast and multicast transmissions.
— Use TCP protocol in data transmissions.
— Object collections processing.
— Data validation.

2 Theoretical background

2.1 About UDP

UDP[3] is the User Datagram Protocol. UDP is designed to send datagrams. Datagrams can be described as discrete blocks of data or messages with limited overhead. UDP does not guarantee that the datagrams will be delivered in any specific order or even at all. They are very useful for certain types of data that does not need 100 reliability, and therefore it does not need the overhead that TCP imposes. Thus it may save time and cost for those specific actions.

2.2 About TCP

TCP[1] is one of the main protocols in TCP/IP networks. Whereas the IP protocol deals only with packets, TCP enables two hosts to establish a connection and exchange streams of data. TCP guarantees delivery of data and also guarantees that packets will be delivered in the same order in which they were sent.

2.3 Unicast vs. Multicast

Unicast[2] is the term used to describe communication where a piece of information is sent from one point to another point. In this case there is just one sender, and one receiver.

Multicast [2] is the term used to describe communication where a piece of information is sent from one or more points to a set of other points. In this case there is may be one or more senders, and the information is distributed to a set of receivers (theer may be no receivers, or any other number of receivers).

One example of an application which may use multicast is a video server sending out networked TV channels. Simultaneous delivery of high quality video to each of a large number of delivery platforms will exhaust the capability of even a high bandwidth network with a powerful video clip server. This poses a major salability issue for applications which required sustained high bandwidth. One way to significantly ease scaling to larger groups of clients is to employ multicast networking.

2.4 Who is this Maven?

The great majority of Maven[4] users are going to call Maven a build tool: a tool used to build deployable artifacts from source code. Build engineers and project managers might refer to Maven as something more comprehensive: a project management tool. What is the difference? A build tool such as Ant is focused solely on preprocessing, compilation, packaging, testing, and distribution. A project management tool such as Maven provides a superset of features found in a build tool. In addition to providing build capabilities, Maven can also run reports, generate a web site, and facilitate communication among members of a working team.

Maven is a build tool, a project management tool, an abstract container for running build tasks. It is a tool that has shown itself indispensable for projects that graduate beyond the simple and need to start finding consistent ways to manage and build large collections of interdependent modules and libraries which make use of tens or hundreds of third-party components. It is a tool that has removed much of the burden of 3rd party dependency management from the daily work schedule of millions of engineers, and it has enabled many organizations to evolve beyond the toil and struggle of build management into a new phase where the effort required to build and maintain software is no longer a limiting factor in software design.

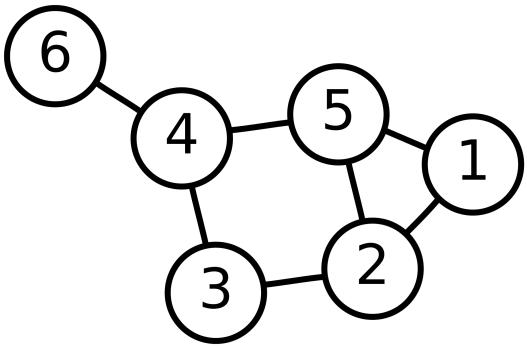

Figure 1: Graph Ilustration

In above illustration is shown a graph, each vertex representing a node in a network. Depending on what data each contain, each can be the Maven. At first look it might appear that 5 or 2 could be the maven because they have most connections. But the process of decision is quite deeper.

3 Solution Descritption

3.1 UDP implementation

A user datagram has a fixed 8 byte header. The header is very simple and contains four, 16 bit fields. The fields are the source port, destination port, total length, and a checksum.

3.1.1 Node.js Implementation

To get started we are going to use the `dgram` class in Node.js. One particular thing to notice is that it doesn't matter whether one is sending or receiving datagrams, you must still bind the socket. This is because datagram sockets are connectionless. Every sender is also a receiver by default and can receive messages.

```
1  var clientUDP = dgram.createSocket('udp4');
2
3  var SRC_PORT = 6025;
4  var PORT = 3000;
5  var MULTICAST_ADDR = '239.255.255.250';
6
7  clientUDP.bind(SRC_PORT, function () {
8      multicastNew();
9      setTimeout(mavenRequest, 2000);
10 });
11
12 clientUDP.on('listening', function () {
13     var address = clientUDP.address();
14     console.log('UDP Client listening on ' +
15             address.address + ':' +
16             address.port +
17             '(multicast address).\n');
18     console.log('Nodes answers:');
19 });
20
21 clientUDP.on('message', function (messageFromNodes, rinfo)...
```

Listing 1: UDP implementation in Node.js

Start out by calling the dgrambind() method to bind the socket to port SRC_PORT on any available IPv4 Ethernet devices. Once the socket has been bound we can listen for any incoming datagram packets by registering an onMessage callback after activating the onListen event handler. One thing to note is that the onMessage callback gives us a msg buffer object. To read the actual datagram we must call the message object (with some optional attributes such as length for instance). The Datagram object returned contains the InternetAddress object rinfo with the details of the sender, and the data that was sent.

3.1.2 UDP Sender

It needsneed to bind() the socket, then call the socket.send(buf, offset, length, port, address[, callback]) method. The send method takes the buffer object as expected, also it takes the destination address and port directly. The address and port parameters tell the datagram where to go. Remember that this is a connectionless protocol so each time we want to send data we need to provide a destination.

```
1  function multicastNew() {
2      var messageClient = new Buffer ('host & neighbours &
           employers');
3      console.log('Sending multicast the command' +
4          messageClient +
5          ' \n to all listening nodes on multicast address ' +
6          MULTICAST_ADDR + ':' + PORT);
```

```
7       clientUDP.send(messageClient, 0, messageClient.length, PORT ,
            MULTICAST_ADDR, function () {
8       });
9   };
```

Listing 2: UDP Sender Function

3.1.3 UDP Multicasting

A single UDP socket can be used to send and receive data. The bind() call establishes what port and address we can receive data on, and the send() call allows us to send data to anywhere we want.

```
1   nodeUDP.on('listening', function () {
2     var address = nodeUDP.address();
3     console.log('UDP Node is listening on ' + address.address + ":"
          + address.port);
4   });
5
6   nodeUDP.on('message', function (messageFromClient, rinfo) {
7     console.log('Recived multicast command'+
8         messageFromClient+
9         'from the client address:' +
10        rinfo.address + ':' + rinfo.port );
```

Listing 3: UDP Multicast Implementation

In the above example, each time a Datagram is received, it is checked whether the message from the sender (client) is of a specific format and based on some logic operations, a result is sent back to the sender. The Datagram object carries the source InternetAddress, and source port that we can use in the send() method to return the message.

Multicasting opens us up to have a single source and multiple destinations. This is very convenient for certain applications like streaming media, or in the laboratory model, the distribution of a collection of JSON objects. The source program(client) sends datagram packets to a multicast group address (239.255.255.250). Each interested client(node) then joins the multicast group and can receive the datagrams being sent.

The source that is sending the multicast datagrams has an easy time, all that is necessary is to send the packets to a multicast group address instead of a normal destination address. Multicast addresses are in the range of 224.0.0.0/4. That is all IP address from 224.0.0.0 to 239.255.255.255.

3.2 TCP Data Transmission

Using TCP is recommended when data integrity is a priority. TCP checks the data and resends the missing packets in case of information losing.

After the Maven is chosen, a TCP transmission is set from node to client and from node to node, as well.

```
1   net.createServer(function(socket) {
2     socket = new JsonSocket(socket);
3     socket.on('message', function(message) {
4       if (message.command == 'RequestData') {
5         socket.sendMessage(employee);
6         ...
```

```
7   }
```

Listing 4: TCP Socket initialization

Sending data is done as shown in next code listing:

```
1  function sendDataToNeighbours(i) {
2    var socket = new JsonSocket(new net.Socket());
3    socket.connect(PORT, neighbours[i], function(){
4      socket.sendMessage({command: 'RequestData'});
5      socket.on('message', function(RequestData) {
6        extend(employee, RequestData);
7      });
8    });
9  };
```

Listing 5: TCP Socket initialization

3.3 Data Collection Processing

Proccessing is happening in some steps. After data is collected, it needs to be valitated.

In this applications a XSD Validation schema was used to do so.

```
1  var xsd = '<xs:schema attributeFormDefault="unqualified"
       elementFormDefault="qualified" xmlns:xs="http://www.w3.org
       /2001/XMLSchema"><xs:element name="MAVEN"><xs:complexType><xs
       :sequence><xs:element name="item" maxOccurs="unbounded"
       minOccurs="0"><xs:complexType><xs:sequence><xs:element type="
       xs:string" name="firstName"/><xs:element type="xs:string"
       name="lastName"/><xs:element type="xs:string" name="
       departament"/><xs:element type="xs:short" name="salary"/></xs
       :sequence></xs:complexType></xs:element></xs:sequence></xs:
       complexType></xs:element></xs:schema>';
```

Listing 6: XSD Validation Schema

Using builtin validation functions for verification we can easily test if a XML is valid or not by its schema definition.

3.4 Workflow

The following steps are done during application testing:

1. Nodes are run by following command:

```
1  //   name    host address    neighbours       its data
2  node node.js 127.0.0.5 3000 '["127.0.0.2" ]' data_5.json
```

Listing 7: Node running

2. The client.js is run by typing `node client.js`.
3. The Client scans the network and choses the Maven.
4. The Maven connects to its neighbors and gets needed data.
5. The Client receives data.
6. **The Client processed verifies using XSD Schema the data collection.** and saves it if test passes.

4 Conclusion

During this laboratory work I gained practical skills in using UDP and TCP transmissions in Node.js.

When it comes to big data collections it is very important to chose correctly which protocol to use and when.

Data manipulation and transmission is very specific to each situation and application. Depending on the business model and priorities, things could be improved a lot by smart choices.

Validation of data is very important and a very useful tool. It can be using in different purposes: verifying integrity, secure input validation etc.

References

[1] Vinton Cerf Carl Sunshine Yogen Dalal. *SPECIFICATION OF INTERNET TRANSMISSION CONTROL PROGRAM*. 1974. URL: https://tools.ietf.org/html/rfc675.

[2] Gorry Fairhurst. *Unicast, Broadcast, Multicast*. 2009. URL: http://www.erg.abdn.ac.uk/users/gorry/course/intro-pages/uni-b-mcast.html.

[3] J. Postel. *User Datagram Protocol*. 1980. URL: https://tools.ietf.org/html/rfc768.

[4] Tim O'Brien Manfred Moser John Casey Brian Fox Jason Van Zyl Eric Redmond Larry Shatzer. *Maven: The Complete Reference*. 2015. URL: http://books.sonatype.com/mvnref-book/reference/index.html.