Seagull - Diameter protocol

Table of contents

1. Diameter protocol details

The implementation of Diameter in Seagull conforms to RFC 3588 (http://www.faqs.org/rfcs/rfc3588.html) . Other applications, like 3GPP/IMS applications (Cx, Dx, Ro, Rf, Sh, ...) are available through Seagull dictionaries.

Note:
Seagull comes with Diameter base and 3GPP/Cx. Including new applications in Seagull is a matter of editing a simple XML file.

2. Getting started with Diameter

2.1. First try

So that you can get familiar with Seagull in the context of Diameter, here is an example that will launch one Diameter server (a server expects a message as the first scenario command) and one Diameter client (a client sends a message as the first scenario command). The client and the server will talk to each other using the loopback interface (127.0.0.1). The scenario is the following:

Note:
The CER/CEA exchange will only be done once, at the time of connection establishment. This is described in the init (core.html#scen_init) section.

Note:
This scenario is included when you install Seagull. It is located in /opt/seagull/diameter/ directory.

Open two terminal sessions. Terminal 2 will be the server and Terminal 1 the client. Examples are located in the "run" directory. So the first thing you need to do is to go in this directory (in both terminal windows):

```
cd run
```

In Terminal 2 window type:

```
./start_server.ksh
```

In Terminal 1 window type:

```
./start_client.ksh
```

On Terminal 2 (server side), you will see:

```
|-------------------------+-----------------------------+-------------------------|
| Start/Current Time      |       2005-12-14 10:04:11 |     2005-12-14 10:06:53 |
|-------------------------+-----------------------------+-------------------------|
|    Counter Name         |      Periodic value         |     Cumulative value    |
|-------------------------+-----------------------------+-------------------------|
| Elapsed Time            | 00:00:01:008                | 00:02:41:596            |
| Call rate (/s)          |   75.397                    |   41.505                |
|-------------------------+-----------------------------+-------------------------|
| Incoming calls          |       76                    |     6707                |
| Outgoing calls          |        0                    |        0                |
| Msg Recv/s              |  149.802                    |   82.985                |
| Msg Sent/s              |  149.802                    |   82.979                |
| Unexpected msg          |        0                    |        0                |
| Current calls           |        3                    |    0.019                |
|-------------------------+-----------------------------+-------------------------|
| Successful calls        |       75                    |     6704                |
| Failed calls            |        0                    |        0                |
| Refused calls           |        0                    |        0                |
| Aborted calls           |        0                    |        0                |
| Timeout calls           |        0                    |        0                |
|-------------------------+-----------------------------+-------------------------|
| Last Info               | Incomming traffic                                     |
| Last Error              | No error                                              |
|--- Next screen : Press key 1 ----------------------- [h]: Display help ------|
```

If you have Ethereal (http://www.ethereal.com/) tool that is started to monitor local (lo) interface, then you should see the Diameter traffic.

Note:

Ethereal must be at least version 0.10.13 to properly decode Diameter.

```
No.     Time        Source      Destination Protocol Info
      4 1.003544    127.0.0.1   127.0.0.1   Diameter Capabilities-Exchange-Request
app=None (hop-id=0) (end-id=0) RPE=100
      6 1.011528    127.0.0.1   127.0.0.1   Diameter Capabilities-Exchange-Answer
app=None (hop-id=0) (end-id=0) RPE=000
      8 2.013175    127.0.0.1   127.0.0.1   Diameter Server-Assignment-Request
app=IMS_Cx_Dx (hop-id=1001) (end-id=2001) RPE=100
      9 2.013760    127.0.0.1   127.0.0.1   Diameter Server-Assignment-Answer
app=IMS_Cx_Dx (hop-id=1001) (end-id=2001) RPE=000
     11 2.014333    127.0.0.1   127.0.0.1   Diameter Server-Assignment-Request
app=IMS_Cx_Dx (hop-id=1002) (end-id=2002) RPE=100
     12 2.014854    127.0.0.1   127.0.0.1   Diameter Server-Assignment-Answer
app=IMS_Cx_Dx (hop-id=1002) (end-id=2002) RPE=000
     13 2.015222    127.0.0.1   127.0.0.1   Diameter Server-Assignment-Request
app=IMS_Cx_Dx (hop-id=1003) (end-id=2003) RPE=100
     14 2.015731    127.0.0.1   127.0.0.1   Diameter Server-Assignment-Answer
app=IMS_Cx_Dx (hop-id=1003) (end-id=2003) RPE=000
```

If you don't have Ethereal, you can take a look at Seagull's log files, which also contain the decoded Diameter messages if Seagull is started with "M" log level (`-llevel ETM`). By default, those files are respectively `client.date.log` and `server.date.log`, suffixed with the date and time at which traffic started.

How easy was that? Now let's jump to the next section to learn how all that works.

2.2. First try explained

Here is the script (start_client.ksh) that launched the client in our example:

```
#!/bin/ksh
```

```
export LD_LIBRARY_PATH=/usr/local/bin

seagull -conf ../config/conf.client.xml -dico ../config/base_cx.xml
-scen ../scenario/sar-saa.client.xml -log ../logs/sar-saa.client.log -llevel ET
```

Note:
On some systems, you might need to include the following export in your Seagull script: "export SHLIB_PATH=/usr/local/bin".

Our example is based on one client that takes care of sending SAR and receiving SAA messages and one server that takes care of receiving SAR and answering SAA messages.

Both sides are relying on the Diameter Base/Cx dictionary provided with Seagull: base_cx.xml to encode Diameter messages. Refer to dictionary configuration section for more information on the format of this dictionary. The dictionary is specified using the `-dico` parameter on the command line (core.html#cli_help) .

The generic configuration (including network and other parameters) is different for the client and the server. The client uses `conf.client.xml` and the server uses `conf.server.xml`. The configuration file is specified using the `-conf` parameter on the command line (core.html#cli_help) .

Here are both files:

conf.client.xml

```
<?xml version="1.0"
encoding="ISO-8859-1"?>
<configuration name="Simple IP
Client Conf">

  <define entity="transport"
    name="trans-ip-v4"
    file="libtrans_ip.so"
create_function="create_cipio_instanc
delete_function="delete_cipio_instanc
    init-args="type=tcp">
  </define>

  <define entity="channel"
    name="trans-ip-v4"
    protocol="diameter-v1"
    transport="trans-ip-v4"
    open-args="mode=client;dest=192.1
  </define>

  <define entity="traffic-param"
          name="call-rate"
          value="1">
  </define>

  <define entity="traffic-param"
          name=""
          value="true">
  </define>

  <define entity="traffic-param"
name="display-protocol-stat"
          value="true">
  </define>

  <define entity="traffic-param"
```

conf.server.xml

```
<?xml version="1.0"
encoding="ISO-8859-1"?>

<configuration name="Simple IP
Server Conf">
  <define entity="transport"
    name="trans-ip-v4"
    file="libtrans_ip.so"
create_function="create_cipio_instance"
delete_function="delete_cipio_instance"
    init-args="type=tcp">
  </define>

  <define entity="channel"
    name="trans-ip-v4"
    protocol="diameter-v1"
    transport="trans-ip-v4"
    open-args="mode=server;source=192.168.0.13:3868">
  </define>

  <define entity="traffic-param"
name="display-scenario-stat"
          value="true">
  </define>

  <define entity="traffic-param"
name="display-protocol-stat"
          value="true">
  </define>

  <define entity="traffic-param"
name="log-protocol-stat-period"
          value="5">
  </define>

  <define entity="traffic-param"
```

```
name="log-protocol-stat-period"
          value="5">
  </define>

  <define entity="traffic-param"
name="log-protocol-stat-name"
          value="all">
  </define>

  <define entity="traffic-param"
name="log-protocol-stat-file"
value="../logs/client-protocol-stat.c
  </define>

  <define entity="traffic-param"
          name="display-period"
          value="1">
  </define>

  <define entity="traffic-param"
          name="log-stat-period"
          value="1">
  </define>

  <define entity="traffic-param"
          name="log-stat-file"
value="../logs/client-stat.csv">
  </define>

  <define entity="traffic-param"
          name="data-log-file"
value="../logs/client-rtt.csv">
  </define>

  <define entity="traffic-param"
          name="call-timeout-ms"
          value="2000">
  </define>

</configuration>
```

```
name="log-protocol-stat-name"
          value="all">
  </define>

  <define entity="traffic-param"
name="log-protocol-stat-file"
value="../logs/server-protocol-stat.csv">
  </define>

  <define entity="traffic-param"
          name="display-period"
          value="1">
  </define>

  <define entity="traffic-param"
          name="log-stat-period"
          value="1">
  </define>

  <define entity="traffic-param"
          name="log-stat-file"
value="../logs/server-stat.csv">
  </define>

</configuration>
```

Table 1: Example client and server configuration

As you can see, the only practical differences between a server and a client are the "mode" (that can be either server or client) in the `open` scenario command and the call-rate parameter which is only specified on the client side.

Note:

Refer to generic configuration (core.html#config_generic) section for more information on the format and possible values.

Now comes the real stuff: the scenario.

First, the scenario source: sar-saa.client.xml (sar-saa.client.xml.html)

And now the commented version:

Scenario	Comments

`<?xml version="1.0" encoding="ISO-8859-1" ?>`
`<scenario>`
 	
`<counter>`
`  <counterdef name="HbH-counter"`
	XML header
	
	Counters definition

```
init="1000"> </counterdef>
  <counterdef name="EtE-counter"
init="2000"> </counterdef>
  <counterdef name="session-counter"
init="0"> </counterdef>
</counter>

<init>
  <send channel="trans-ip-v4">
    <command name="CER">
      <avp name="Origin-Host"
value="seagull.ims.hpintelco.org">
</avp>
      <avp name="Origin-Realm"
value="ims.hpintelco.org"> </avp>
      <avp name="Host-IP-Address"
value="0x00010a03fc5e"> </avp>
      <avp name="Vendor-Id"
value="11"> </avp>
      <avp name="Product-Name"
value="HP Cx Interface"> </avp>
      <avp name="Origin-State-Id"
value="1094807040"> </avp>
      <avp
name="Supported-Vendor-Id"
value="10415"> </avp>
      <avp
name="Auth-Application-Id"
value="167772151"> </avp>
      <avp
name="Acct-Application-Id"
value="0"> </avp>
      <avp
name="Vendor-Specific-Application-Id"
        <avp name="Vendor-Id"
value="11"></avp>
        <avp
name="Auth-Application-Id"
value="167772151"></avp>
        <avp
name="Acct-Application-Id"
value="0"></avp>
      </avp>
      <avp name="Firmware-Revision"
value="1"> </avp>
    </command>
  </send>

  <receive channel="trans-ip-v4">
    <command name="CEA">
    </command>
  </receive>
</init>

<!-- Traffic -->
<traffic>
  <send channel="trans-ip-v4">
    <action>
      <!-- For each new call,
increment the session-ID counter -->
      <inc-counter
name="HbH-counter"> </inc-counter>
      <inc-counter
name="EtE-counter"> </inc-counter>
      <inc-counter
name="session-counter">
```

```
The init scenario, executed only
once when Seagull is started
Send on "trans-ip-v4" channel as
defined in config file
Send CER command as defined in
dictionary file
List of Diameter avps

The Host-IP-Address is consituted of
IP address type
(0001 for IPV4) and the IP address
in hex form

Grouped AVP
"Vendor-Specific-Application-Id"

Wait for the CEA

End of the init scenario

Beginning of the traffic scenario

List of actions to execute before
sending the message
(before "command")
Increment Hop-by-Hop counter
Increment End-to-End counter
Increment session-counter counter
Set the value of the Hop-by-Hop
field in Diameter header

Set the value of the End-to-End
field in Diameter header

Set the value of the Session-Id avp
The "session-counter" value is put
in place of $(session-counter)

Send the SAR message
Value of Session-Id AVPs is set by
the previous "action"
```

```
</inc-counter>
      <set-value name="HbH-id"
format="$(HbH-counter)"></set-value>
      <set-value name="EtE-id"
format="$(EtE-counter)"></set-value>
      <set-value name="Session-Id"
format=".;1096298391;$(session-counte
    </action>
    <command name="SAR">
      <avp name="Session-Id"
value="value_is_replaced"> </avp>
      <avp
name="Vendor-Specific-Application-Id"
        <avp name="Vendor-Id"
value="11"></avp>
        <avp
name="Auth-Application-Id"
value="167772151"></avp>
        <avp
name="Acct-Application-Id"
value="0"></avp>
      </avp>
      <avp name="Auth-Session-State"
value="1"> </avp>
      <avp name="Origin-Host"
value="seagull"> </avp>
      <avp name="Origin-Realm"
value="ims.hpintelco.org"> </avp>
      <avp name="Destination-Realm"
value="ims.hpintelco.org"> </avp>
      <avp name="Server-Name"
value="seagull"> </avp>
      <avp
name="Server-Assignment-Type"
value="3"> </avp>
      <avp
name="User-Data-Request-Type"
value="0"> </avp>
      <avp name="Public-Identity"
value="sip:olivierj@ims.hpintelco.org
</avp>
      <avp name="Destination-Host"
value="hss.ims.hpintelco.org">
</avp>
    </command>
    <action>
      <start-timer></start-timer>
    </action>
  </send>

  <receive channel="trans-ip-v4">
    <action>
      <stop-timer></stop-timer>
    </action>
    <command name="SAA">
    </command>
  </receive>
</traffic>

</scenario>
```

```
Start the timer to measure response
time


Receive the SAA message on
"trans-ip-v4" channel

End the timer when SAA has been
received




End of traffic description

End of scenario
```

In this example, the `init` section takes care of sending a CER and receiving a CEA. You can put any <avp> described in the `base_cx.xml` dictionary. The `traffic` section keeps sending SAR/SAA messages.

3. Generic configuration

The generic configuration file describes the network environment as well as traffic parameters.

The network environment is described through "transport channel entities (core.html#ref_transport) ". The transport entity is then used as an attribute of `send` (core.html#cmd_send) and `receive` (core.html#cmd_receive) scenario commands, as well as during the opening of the transport channel (see below).

```
<!-- DIAMETER example -->
<?xml version="1.0" encoding="ISO-8859-1"?>
<configuration name="Simple Client Conf">

<define entity="transport"
    name="trans-ip-v4"
    file="libtrans_ip.so"
    create_function="create_cipio_instance"
    delete_function="delete_cipio_instance"
    init-args="type=tcp">
</define>
<!-- Then you specify the opening of the channel, on the transport previously
described. -->

<!-- For a server listening to port 3868 on interface "192.168.0.13", it will look like
this: -->

<define entity="channel"
    name="trans-ip-v4"
    protocol="diameter-v1"
    transport="trans-ip-v4"
    open-args="mode=server;source=192.168.0.13:3868">
</define>

<!-- For a client sending messages to port 3868 on interface "192.168.0.13", it will
look like this: -->
<define entity="channel"
    name="trans-ip-v4"
    protocol="diameter-v1"
    transport="trans-ip-v4"
    open-args="mode=client;dest=192.168.0.13:3868">
</define>
```

4. Diameter dictionary

Diameter is constituted of a base protocol and additional "applications". In order for Seagull to be versatile enough, Diameter messages and parameters are described in an XML dictionary. Seagull comes with a complete Diameter base and Cx interfaces dictionary.

A dictionary contains several XML sections

4.1. Types

"types" section contains all types needed for the protocol. For Diameter, these are:

```
<types>
  <!-- Types defined for the AVP   -->
  <typedef name="Integer32" type="signed" size="4" unit="octet"> </typedef>
  <typedef name="Unsigned32" type="number" size="4" unit="octet"> </typedef>
  <typedef name="Integer64" type="signed64" size="8" unit="octet"> </typedef>
```

```
  <typedef name="Unsigned64" type="number64" size="8" unit="octet"> </typedef>
  <typedef name="OctetString" type="string" unit="octet"> </typedef>
  <typedef name="Grouped" type="grouped"></typedef>
</types>
```

4.2. Header

"header" section contains the description of message header. For Diameter, this is:

```
<header name="command" length="msg-length" type="cmd-code">
  <fielddef name="protocol-version" size="1" unit="octet"></fielddef>
  <fielddef name="msg-length" size="3" unit="octet"> </fielddef>
  <fielddef name="flags" size="1" unit="octet"> </fielddef>
  <fielddef name="cmd-code" size="3" unit="octet"> </fielddef>
  <fielddef name="application-id" size="4" unit="octet"> </fielddef>
  <fielddef name="HbH-id" size="4" unit="octet"> </fielddef>
  <fielddef name="EtE-id" size="4" unit="octet"> </fielddef>
</header>
```

4.3. Body

"body" section contains the description of message body (which naturally comes after the header). For Diameter, this is:

```
<body>
  <header name="avp" length="avp-length" type="avp-code">
     <fielddef name="avp-code" size="4" unit="octet"> </fielddef>
     <fielddef name="flags" size="1" unit="octet"> </fielddef>
     <fielddef name="avp-length" size="3" unit="octet"> </fielddef>
     <optional>
        <fielddef name="Vendor-ID" size="4" unit="octet"
                  condition="mask" field="flags" mask="128">
        </fielddef>
     </optional>
  </header>
</body>
```

4.4. Dictionary

"dictionary" section contains all possible parameters that a message can contain. Here is a description for some Diameter AVPs:

```
<dictionary>
  <avp>
    <!-- Diameter base AVPs -->
    <define name="User-Name" type="OctetString">
      <setfield name="avp-code" value="1"></setfield>
      <setfield name="flags" value="64"></setfield>
    </define>
    <define name="Host-IP-Address" type="OctetString">
      <setfield name="avp-code" value="257"></setfield>
      <setfield name="flags" value="64"></setfield>
    </define>
    <define name="Auth-Application-Id" type="Unsigned32">
      <setfield name="avp-code" value="258"></setfield>
      <setfield name="flags" value="64"></setfield>
    </define>
  </avp>
</dictionary>
```

4.5. Command

"command" section contains all possible Diameter messages (called command in RFC 3588). Here is an example of Diameter messages:

```
<define name="SAR">
   <!-- It's a request, R bit is set -->
   <setfield name="flags" value="128"> </setfield>
   <setfield name="cmd-code" value="301"></setfield>
   <setfield name="application-id" value="167772151"></setfield>
   <setfield name="protocol-version" value="1"></setfield>
</define>
<define name="SAA">
   <!-- It's an answer, R bit is unset -->
   <setfield name="flags" value="0"> </setfield>
   <setfield name="cmd-code" value="301"></setfield>
   <setfield name="application-id" value="167772151"></setfield>
   <setfield name="protocol-version" value="1"></setfield>
</define>
```

5. Actions in commands for Diameter

The <send> and <receive> scenario commands include an <action> and <command> section.

The <action> section can be placed before or after the <command> section.

Actions placed before the command (called "**pre-actions**") are executed before the message is actually sent or received. Actions placed after the command (called "**post-actions**") are executed after the message is sent or received.

In the context of Diameter, the following actions are used to maintain Diameter Hop-by-Hop ID, End-to-End ID and Session ID. Example:

```
  <send channel="trans-ip-v4">
         <action>
           <!-- For each new call, increment the session-ID counter -->
           <inc-counter name="HbH-counter"> </inc-counter>
           <inc-counter name="EtE-counter"> </inc-counter>
           <inc-counter name="session-counter"> </inc-counter>
           <set-value name="HbH-id"
                      format="$(HbH-counter)"></set-value>
           <set-value name="EtE-id"
                      format="$(EtE-counter)"></set-value>
           <set-value name="Session-Id"
                      format=".;1096298391;$(session-counter)"></set-value>
         </action>
```

Actions that can be placed after a command are actions to retrieve an AVP value after the message has been received. Example:

```
  <receive channel="trans-ip-v4">
         <command name="SAR">
           <!-- Only need to specify what needs to be parsed -->
           <avp name="Session-Id" value="dont_care"> </avp>
         </command>
         <!-- Store action is at the end of the receive command -->
         <action>
           <store name="HbH" entity="HbH-id"></store>
           <store name="E2E" entity="EtE-id"></store>
           <store name="sid" entity="Session-Id"></store>
         </action>
       </receive>
```

The list of possible actions (core.html#ref_actions) is available in the reference section.