StrongLeaf SMPPRouter Design Document

by

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME
1	September 2015	Initial SMPPRouter documentation.	HHPF

Contents

1 Introduction **1**

2 Features **1**

3 Current Limitations and Assumptions **1**

4 Requirements for deployment **1**

5 Implementation **1**

5.1 Overview . . . 1
5.2 Data Model . . . 1
5.3 Data Model DIDManager . . . 2
5.4 Performance . . . 2
5.5 Load-sharing and Fail-over . . . 2
5.6 Message and Error handling . . . 2
5.6.1 DIDManager . . . 2
5.7 Routing . . . 3
5.8 Patcher . . . 3
5.8.1 GSM 7-bit patcher . . . 3

6 Configuration using Smalltalk **3**

6.1 SMPPRouter with systemID routing . . . 3
6.2 SMPPRouter as DIDManager . . . 4
6.3 Configuring a patcher . . . 5
6.4 REST Interface . . . 6

7 Monitoring **6**

7.1 Statistics . . . 6

8 Management **6**

8.1 Customer Management . . . 6
8.2 MSISDN Mapping Management . . . 7

1 Introduction

The SMPPRouter will route SubmitSM and DeliverSM messages based on a configurable criteria between different links. It can be configured to multiple incoming and outgoing connections (and act either as ESME or MC) and each of them is associated with the SMPP systemId. For each incoming message the outgoing path will be determined. Before forwarding the message various parts of the message and header can be patched.

The extension to the SMPPRouter is the DIDManager that will look up the destination number in a database and determine where to forward the message.

2 Features

- SMPPv3.4 support
- SMPP Enquire Link is done locally
- Routing decision by destination number can be done using a database look-up.

3 Current Limitations and Assumptions

- SMPP extensions in messages are not allowed.
- Each incoming connection is associated with a single systemId
- No load-balancing between links with the same systemId.

4 Requirements for deployment

A Debian 64-bit system should be used. For the DIDManager functionality a Mongo database should run on the same system.

Please see best practices for deploying Mongo database systems in terms of RAM, disk space and maintenance.

5 Implementation

5.1 Overview

The SMPPRouter can accept connections or make connections itself. It does not follow the classic ESME/MC split but is happy to send and receive messages on any link.

5.2 Data Model

The SMPPRouter is made up of different components. The core is a router, connections and patchers. A route determines to which systemId a message should be sent and then a matching connection will be found.

SMPPSystemIdRouter
: Route DeliverSM/SubmitSM based on the systemId of the source link.

SMPPSystemIdRoute
: A specific route that solely uses the systemId of the source link.

SMPPSystemIdBodyRoute
: A specific route that is uses the systemId of the source link and regular expressions for the source and destination number.

SMPPNumberPatcher
Remove a fixed prefix from the destination MSISDN in a DeliverSM message.

SMPP7BitPackingPatcher
A SMPP message re-write element that will pack a message for a configurable DCS to create the message of a GSM 7-bit encoded message.

SMPPConnection
An outgoing connection to a remote SMPP system. The SMPPRouter would send DeliverSM/SubmitSM on such a link ignoring the rules of ESME/MC.

SMPPConnectionManager
Wait for an incoming system. The SMPPRouter would send DeliverSM/SubmitSM on such a link ignoring the rules of ESME/MC. Multiple incoming connections can be active at the same time.

5.3 Data Model DIDManager

SMPPSystemIdMongoRouter
Route based on the destination number

SMPPMongoRoute
A specific route that matches a destination number and defines the system it should be sent to.

5.4 Performance

Provide benchmark results.

5.5 Load-sharing and Fail-over

One can create multiple connections with the same systemId. The SMPPRouter will pick the first connection that is connected when routing. The response will always be sent through the originating connection. Right now there is no load-sharing but fail-over can be achieved by creating several connections with the same systemId. If the source connection fails during a pending DeliverSM/SubmitSM the result will not be routed through an alternative.

5.6 Message and Error handling

This is the standard behavior for all SMPPRouters

Nr.	Condition	Action
PR01	Message filtered	No response
PR02	No route	SMPP Command Status 8
PR03	Cancel	SMPP Command Status 8
PR04	Error	Forward error
PR05	Success	Forward success
PR06	Timeout	No response

5.6.1 DIDManager

These are the specific handling conditions for the DIDManager.

Nr.	Condition	Action	Reason
PR02	No route	Send success	Unrouted numbers should not be retried.

5.7 Routing

The SMPPSystemIdRouter will search rules in the order they were added. So in case two rules match the rule that was added first will match. The route selection will not check if the routed to systemId has a working connection.

5.8 Patcher

The patchers can be used to manipulate DeliverSM/SubmitSM messages before them being forwarded. This can be used for general routing re-writing or catering for specific quirks of remote systems.

There can be one global patcher and a list of named patchers that will activate depending on the route that was selected. In case a requested patcher is not available no patching will occur and no error is generated.

Each patcher can be assigned to have patcherName that will be used by the patcher selection.

5.8.1 GSM 7-bit patcher

A 7-bit SMS will be transported using the DCS=0 and it will be unpacked. This means that each septet will be stored in a single octet (leaving one bit unused). Some equipment might not be able to deal with this.

The patcher can be configured to match a specific DCS and will then pack the SMPP short_message field.

6 Configuration using Smalltalk

6.1 SMPPRouter with systemID routing

```
| router clientConnection1 clientConnection2 serverConnection1 serverConnection2 syslog  ↩
    statsd queuedStatsd |

syslog := LogTargetSyslog openlog: 'smppRouter' option: 0
        facility: LogTargetSyslog LOG_USER.
syslog prefix: ''.
LogManager default target: syslog.

clientConnection1 := SMPPConnection new
                     hostname: '172.16.1.81';
                     port: 2775;
                     systemId: 'smscMC';
                     password: 'PW1';
                     systemType: 'GSM';
                     yourself.
clientConnection2 := SMPPConnection new
                     hostname: '127.0.0.1';
                     port: 2776;
                     systemId: 'homeMC';
                     password: 'PW2';
                     systemType: 'GSM';
                     yourself.
serverConnection1 := SMPPConnectionManager new
                     password: 'PW3';
                     systemId: 'smscESME';
                     systemType: 'GSM';
                     yourself.
serverConnection2 := SMPPConnectionManager new
                     password: 'PW4';
                     systemId: 'homeESME';
                     systemType: 'GSM';
```

```
                    yourself.

"Select the kind of router to use. Add connections and routes"
router := SMPPSystemIdRouter new
           addConnection: clientConnection1;
           addConnection: clientConnection2;
           addConnection: serverConnection1;
           addConnection: serverConnection2;
           routeSystemId: 'smscMC' sourceAddress: '[0-9]*' asRegex
                destinationAddress: '[0-9]*' asRegex toSystemId: 'homeESME';
           routeSystemId: 'smscESME' sourceAddress: '49123[0-9]*' asRegex
                destinationAddress: '[0-9]*' asRegex toSystemId: 'homeMC';
           routeSystemId: 'homeMC' sourceAddress: '[0-9a-zA-Z]*' asRegex
                destinationAddress: '49456[0-9]*' asRegex toSystemId: 'smscESME';
           yourself.

"Optional statsD support"
statsd := UDPStatsDClient new.
statsd hostname: 'statsdserver'.
statsd port: 1234.
statsd start.

"Do not send each stat directly but queue it"
queuedStatsd := QueuedStatsDClient new.
queuedStatsd client: statsd.

"Now tell the SMPPRouter to use it"
router statsClient: queuedStatsd.


"Connect and wait for connections"
clientConnection1 start.
clientConnection2 start.
serverConnection1 start: 2775.
serverConnection2 start: 2770.
```

6.2 SMPPRouter as DIDManager

```
| database router clientConnection1 clientConnection2 serverConnection1 serverConnection2  ↩
    syslog statsd queuedStatsd |

syslog := LogTargetSyslog openlog: 'smppRouter' option: 0
        facility: LogTargetSyslog LOG_USER.
syslog prefix: ''.
LogManager default target: syslog.

clientConnection1 := SMPPConnection new
                     hostname: '172.16.1.81';
                     port: 2775;
                     systemId: 'smscMC';
                     password: 'PW1';
                     systemType: 'GSM';
                     yourself.
clientConnection2 := SMPPConnection new
                     hostname: '127.0.0.1';
                     port: 2776;
                     systemId: 'homeMC';
                     password: 'PW2';
                     systemType: 'GSM';
                     yourself.
```

```
serverConnection1 := SMPPConnectionManager new
                     password: 'PW3';
                     systemId: 'smscESME';
                     systemType: 'GSM';
                     yourself.
serverConnection2 := SMPPConnectionManager new
                     password: 'PW4';
                     systemId: 'homeESME';
                     systemType: 'GSM';
                     yourself.

database := VOMongoRepository database: 'adbName'.

"Select the kind of router to use. Add connections and routes"
router := SMPPSystemIdMongoRouter new
           addConnection: clientConnection1;
           addConnection: clientConnection2;
           addConnection: serverConnection1;
           addConnection: serverConnection2;
           database: database;
           yourself.

"Optional statsD support"
statsd := UDPStatsDClient new.
statsd hostname: 'statsdserver'.
statsd port: 1234.
statsd start.

"Do not send each stat directly but queue it"
queuedStatsd := QueuedStatsDClient new.
queuedStatsd client: statsd.

"Now tell the SMPPRouter to use it"
router statsClient: queuedStatsd.


"Connect and wait for connections"
clientConnection1 start.
clientConnection2 start.
serverConnection1 start: 2775.
serverConnection2 start: 2770.
```

6.3 Configuring a patcher

This assumes that a router has already been created and a new patcher will be created and registered with the system.

```
"Create a patcher"
patcher := SMPP7BitPackingPatcher new.
patcher dcs: 0.
patcher patcherName: 'PackBits'.

"Register the patcher"
router addNamedPatcher: patcher.


"In case the SMPPSystemIdRouter is used. The cascade to
add multiple rules need to be-rewritten to one rule per
line and then set the patches to apply."
route := router routeSystemId: 'homeMC'
                sourceAddress: '[0-9a-zA-Z]*' asRegex
```

```
                destinationAddress: '49456[0-9]*' asRegex toSystemId: 'smscESME'.
route patcherNames: #('PackBits' 'OtherPatch').
```

6.4 REST Interface

This will launch a REST service to manage the customer, sponsor and mapping. The port can be modified and basic authentication can be enabled for the server.

```
| database uriSpace server |
database := VOMongoRepository database: 'did-db'.
uriSpace := SMPPMongoRestUriSpace new.
uriSpace database: database.
server := ZnServer startOn: 1700.
server delegate:
        (ZnJSONRestServerDelegate new
                uriSpace: uriSpace;
                yourself); yourself.
```

7 Monitoring

There is no dedicated REST monitoring interface in this version of the software. The database and syslog can be monitored at this point in time. E.g. the number of allocated states could be checked, the node id could be determined, the amount of the CS/PS IMSI mappings.

When the receiving SMPP process is busy/blocked the system recv queue will grow and this can be monitored using the standard netstat command.

7.1 Statistics

The SMPPRouter is counting several events and is exporting them using

8 Management

8.1 Customer Management

Creating a customer entry

```
$ curl -H "Content-Type: application/json" -XPUT \
http://localhost:1700/v1/customer/Customer \
 -d '{ \
        "systemId": "SysId", \
        "sipProxyIP": "10.2.3.4", \
        "smppPatcherNames": ["PackBits"], \
        "sipProxyPort": 5060}'
```

Getting a customer entry

```
$ curl -H "Content-Type: application/json" -XGET \
http://localhost:1700/v1/customer/Customer
{
        "sipProxyPort" : 5060,
        "systemId" : "SysId",
        "customerName" : "Customer",
        "smppPatcherNames": ["PackBits"],
        "sipProxyIP" : "10.2.3.4"
}
```

8.2 MSISDN Mapping Management

Creating a mapping

```
$ curl -H "Content-Type: application/json" -XPUT \
http://localhost:1700/v1/routing/49123456 \
-d '{"customerName": "Customer"}'
```

Getting a mapping

```
$ curl -H "Content-Type: application/json" -XGET \
http://localhost:1700/v1/routing/49123456
{
        "customerName" : "Customer",
        "msisdn" : "49123456"
}
```

Deleting a mapping

```
$ curl -H "Content-Type: application/json" -XDELETE \
http://localhost:1700/v1/routing/49123456
OK
```