SMPP Message

Load-Balancing

Solution



21 Aug 2018

Version 2

Compiled By:

Vernon Wells

F5 Solution Architect

Updated, 20 May 2020

Jesse Driskill

F5 Systems Engineer

## Overview

This document describes the current SMPP message load-balancing solution as it is implemented on the F5 BIG-IP platform.

## Architecture

The architecture can be described as having two sides, which will arbitrarily be called “left” and “right”. The left side consists of SAG servers, which proxy on behalf of multiple ESMEs (clients). The right side consists of SMSCs (servers). The BIG-IP platform sits between the left and right sides and acts as an SMPP proxy.

The BIG-IP platform MRF connects to peers. Each SAG (ESME) and each SMSC is a peer. A peer that initiates transport toward the BIG-IP platform is called a “dynamic peer”. A peer to which the BIG-IP platform initiates a transport connection is called a “static peer”, in MRF parlance. A peer may act as both a static peer and a dynamic peer as long as the SMSC set it is initiating connections to (as a dynamic peer) is never the same SMSC set in which it is defined as a static peer.

The current architecture treats SMSCs as static peers and SAGs as dynamic peers. It does not need to be this way but is currently constructed in this manner.

Within the BIG-IP configuration SMSCs are grouped by sets, with separate BIG-IP configuration objects and resources for each set. Unique objects associated with each SMSC set include:

* **Data Director virtual-server**
  + Virtual-server that all ESMEs (SMPP clients) connect to. The director vip uses the bind-id provided by the ESME to identifies the data-plane VS that the ESME should be connected to and forwards the connection internally.
* **Management Director virtual-server**
  + Similar to the Data-director, the Management-director forwards SMPP management/control connections to the appropriate management-plane virtual based on the SMPP bind-id.
* **Service-checking virtual-server**
  + Used for health-checking SMSCs and re-binding to any that are available but not connected. This virtual is only used by an iCall script and does not receive external connections
* **Data-plane virtual-server**
  + Handles SMPP data-plane connections from ESMEs
* **Management-plane virtual-server**
  + Handles management/control connections for a given SMSC set.
* **‘smpp\_bind\_map’ data-group**
  + Contains SMPP bind-ids and the SMSC-sets they map to
* **SMSC-set data-group(s)**
  + Data-group containing the objects associated with a unique SMSC set, including the data and control virtual-servers, the pool name, MRF transport-config name, and the credentials for binding to SMSCs
* **Pool**
  + Contains the SMSCs (servers) included in this set. An SMSC may be part of multiple SMSC sets.
* **Message-Routing Framework (MRF) objects**
* Peer, Protocol, Transport, Route, and Router MRF objects for each SMSC set

## SMPP Proxy Initialization

The configuration for an “SMSC set” includes several configuration elements and, due to the complexity of the system, is typically deployed using Ansible. Ansible roles have been written to handle the initialization of an SMPP Proxy, the deployment of SMSC sets, and the removal of SMSC sets.

The ‘initialize\_smpp’ Ansible role is used to initialize the SMPP proxy system on a BIG-IP. This involves the deployment of the following configuration elements:

* ‘data\_director’ virtual-server
* ‘mgmt\_director’ virtual-server
* ‘smsc\_check’ virtual-server
* ‘smpp\_bind\_map’ data-group
* ‘smsc\_check’ data-group
* ‘smpp\_data\_plane’ iRule
* ‘smpp\_mgmt\_plane’ iRule
* ‘smpp\_data\_director’ iRule
* ‘smpp\_mgmt\_director’ iRule
* ‘smpp\_smsc\_check’ iRule
* ‘smsc\_check’ iCall script
* ‘smsc\_check’ iCall handler
* ‘smpp\_tcp’ TCP profile

## SMSC Set Deployment

SMSC sets are deployed with the ‘deploy\_smsc\_set’ Ansible role. All of the tasks in the rule use a variables file to populate the configuration. The only configuration required to deploy a new SMSC-set is a variables file; the Ansible task files themselves never need direct modification (subject to future enhancements or BIG-IP versions). The Ansible role deploys the configuration elements shown below.

Note that the object names are derived from the SMSC-set name. The SMSC-set used for these examples was named ‘esme01, and the portion of the name derived from the SMSC-set name is highlighted red.

* SMSC-set data-group: esme01
* Data-plane virtual: vs01a-data\_esme01
* Mgmt-plane virtual: vs01b-mgmt\_esme01
* SMSC-set pool: esme01-pool
* MRF Protocol: esme01-protocol
* MRF Transport: esme01-transport
* MRF Peer: esme01-peer
* MRF Route: esme01-route
* MRF Router: esme01-router

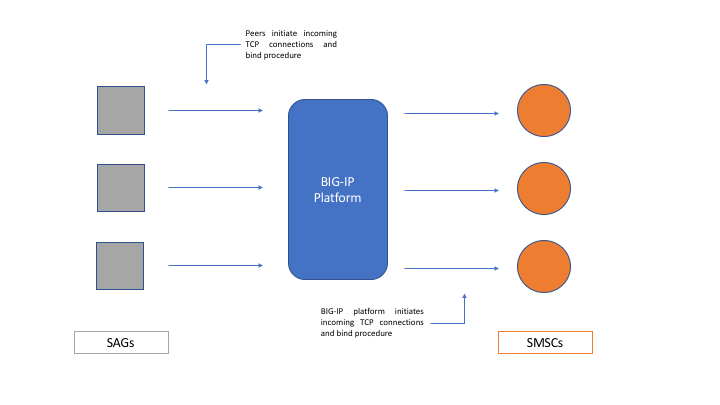
## SMSC Set Removal

SMSC-sets should be removed using the ‘remove\_smsc\_set’ Ansible Role. The role uses the exact same variables file that was used to deploy the SMSC-set. A variable file created to deploy an SMSC-set should be retained in case that SMSC-set must be removed or reconfigured in the future.

## SMPP Operation Overview

An SMPP session is initiated with a bind procedure. A bind may declare the underlying transport as a “receiver” (meaning requests are received by the initiator), “transmitter” (meaning requests are transmitted by the initiator) or “transceiver” (meaning requests may flow either direction). A bind also declares a *system\_id*, which identifies the system to peers, and a *system\_type*, which is an arbitrary label intended to identify something about the nature or source of the messages sent or received over the associated transport.

In the current architecture, the BIG-IP platform is an explicit SMPP proxy. It has its own unique system\_id, which it asserts during bind procedures, and it independently manages transport binding on the left and right side. At this point, the BIG-IP platform expects to initiate the bind procedure to static peers (i.e., SMSCs), and to receive bind requests from dynamic peers (i.e., ESMEs/SAGs).

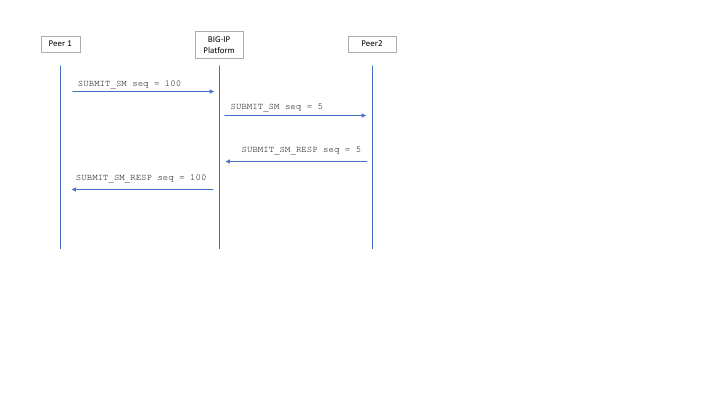


If the BIG-IP receives an unbind request from any peer, it sends an unbind response back to that peer and closes the associated transport.

An instance with a transceiver bind infers whether there are any peers available to receive messages based on whether there are any bound incoming transports with that same asserted bind. Because a proxy hides all other peers from each instance, and the bind transports exist between each instance and the proxy, the proxy must manage bind state based on this implicit inference system. As such, the BIG-IP should not initiate a bind toward an SMSC unless there is at least one ESME that is bound for that system\_type.

## Message Handling

A message from any peer can be routed to any other peer by MRF. SMPP transactions consist of a request message and a response message (except alert\_notification and generic\_nack, which are uni-directional). MRF must rewrite the sequence number of an incoming request message to a value that is scoped and temporally unique for the transport toward the destination peer. The sequence number of a response must also be rewritten to match that used by the original requestor. MRF will use the value of the rewritten sequence number to match responses to requests, in the usual SMPP fashion. This ensures that responses are routed back to the peer which initiated the matching request.



Messages are routed only to transports that are part of the SMSC set that the ESME initially bound to (based on bind\_id). There is no persistence for message flows. The destination for each request message is routed independently of all other request messages from the same peer and bind.

The BIG-IP directly handles enquire\_link and unbind commands initiated by peers. All other message types are routed as described above.

## Considerations

* For request, send transient error (0x64 in status)
* For response, if dropped, do nothing
* No need to validate ESME password
* SV\_SAGF1\_VMC1 [test]

## BIG-IP SMPP Proxy Operation

The *smpp\_bind\_map* data-group associates BIND IDs with SMSC-sets. All SMPP connections from ESMEs connect to either the *data\_director* or the *mgmt\_director* virtual-server and send a BIND\_REQUEST. The associated iRule (*smpp\_data\_director* or *smpp\_mgmt\_director*) uses the bind\_id to find the associated SMSC-set in the smpp\_bind\_map data-group. Once the correct SMSC set is found the iRule then references that data-group to identify the virtual-server that should be used and forwards the client connection. The diagram below illustrates the relationship between the *\*\_director* virtual-servers, the *smpp\_bind\_map* data-group, the SMSC-set data-group, and the actual SMPP proxy virtual-servers.

A picture containing text, map

Description automatically generated

1. The *smpp\_data\_****director*** iRule uses the BIND ID from the BIND\_REQUEST to find the correct SMSC set in the *smpp\_bind\_*map data-group.
2. The *smpp\_data\_****director*** iRule then finds the data-plane virtual server for that SMSC set.
3. The connection is forwarded to the data-plane virtual-server identified from the SMSC-set data-group.
   1. This virtual-server is using and internal address and is not directly accessible from the network.
4. The *smpp\_data\_****plane*** iRule checks the *smpp\_bind\_map* data-group to identify the appropriate SMSC set
5. The *smpp\_data\_****plane*** iRule retrieves required variables from the SMSC-set data-group, such as the SMSC-set name, MRF transport and protocol profiles, bind id/password to use with the peers (SMSCs), etc…[[1]](#footnote-2)

This approach allows all ESMEs to connect to a single virtual-server address while still directing them to discrete sets of SMSCs.

## Management iRule Commands

As noted above, each SMSC set has two associated virtual servers: a data-plane VS and a management-plane VS. The management virtual-server accepts TCP connections and expects a single, unencrypted command to be submitted. A command will consist of a bind ID, a command, and in some cases, command arguments. Commands apply *only* to the SMSC set specified by the bind id. For example, to disconnect all ESMEs connected to every SMSC set, the ‘unbind\_esmes’ command would have to be executed for each SMSC set individually.

The simplest method to send management commands is using netcat:

*echo -n “<bind\_id> <command>” | nc <mgmt\_director\_address> 5555*

Supported commands are as follows:

* enquire <smsc\_address> <smsc\_port>
  + Send link\_enquire to specified SMSC ip:port
* prime\_smscs
  + Prime all SMSCs connected to SMSC-set
* close\_smsc
  + Disconnect a single SMSC
* unbind\_smscs
  + Unbind all SMSCs connected to the SMSC-set
* unbind\_esmes
  + Unbind all ESMEs connected to the SMSC set
* show\_esmes
  + Print a count and list of all ESMEs connected to the SMSC set
* show\_smscs
  + Print count of all connected SMSCs in set
* Show\_bound\_smsc\_count
  + Print the number of SMSCs currently connected for this SMSC set
* reset
  + UNBIND all SMSCs and ESMEs connected to the SMSC set and reset SMSC set statistics

Management connections are made to the public management virtual-server. Like the public data-plane virtual-server, the management virtual identifies to correct virtual-server set based on the bind id. The bind id is provided as the first argument in the command. Below are some examples of using command-line tools to work with the management virtual-server:

* echo -n "**esme01 prime\_smscs**" | netcat 10.1.10.101 5555
  + This command will cause the BIG-IP to connect to all of the SMSCs peers in the pool and send a BIND\_REQUEST to each.
  + *esme01* is the bind id used to direct the command to the appropriate SMSC set
  + *prime\_smscs* is the command to be executed
* echo -n "**esme01 unbind\_esmes**" | netcat 10.1.10.101 5555
  + This command will cause the BIG-IP to send an UNBIND\_REQUEST to every ESME connected to this SMSC set.

## iCall and the ‘smsc\_check’ iRule

During execution the data-plane iRule compares the number of connected SMSCs to the number of enabled SMSCs and initiates a connection to any that are not already connected. As this is part of the iRule, the check only occurs when the iRule is executed, which was considered sub-optimal by the customer. The customer wanted the connected vs. active check to happen automatically as long as ESMEs were connected to the SMSC-set, regardless of whether they were sending any traffic.

To support this requirement a third ‘public’ virtual-server was created to host the *smsc\_check* iRule. The *smsc\_check* iRule runs the same comparison as described above, but it does so for all of the SMSC-sets at once. When an SMSC-set has ESMEs connected, the count of connected SMSCs is compared to the count of enabled SMSCs. If an SMSC is found that is both enabled and not connected, a connection to it is opened using the MR::prime iRule command.

The *smsc\_check* periodic iCall handler is used to run the *smsc\_check* script every ten seconds. The *smsc\_check* iCall script connects to the \*\_*smsc\_check* virtual-server, which executes the *smpp\_smsc\_check* iRule. The configuration for the smsc\_check iCall handler and script is shown below.

sys icall handler periodic smsc\_check {

interval 10

script smsc\_check

}

sys icall script smsc\_check {

app-service none

definition {

set bash\_cmd "nc 10.1.10.101 5556 < <(echo -n active\_smsc\_check)"

if { [catch { exec /bin/bash -c $bash\_cmd} result] } {

puts "Bash command failure"

}

else {

puts $result

}

}

description none

events none

}

## Data-Plane iRule Sections

This section contains a summary of the actions taken by the *smpp\_data\_plane* iRule.

###### RULE\_INIT

* Set $static::esme\_bindings to ‘smpp\_bind\_map’, which contains the mapping of bind\_ids to smsc\_sets
* Define the $static::smpp\_command\_map array, which contains SMPP commands and their numerical IDs

###### CLIENT\_ACCEPTED

* Initialize variables for this ESME connection
* Create the GENERICMESSAGE::peer
* Begin collecting TCP

###### CLIENT\_DATA

* Verify at least 16 bytes of data collected (minimum valid SMPP message size)
* Extract $command\_length, $command\_id, $command\_status, $sequence\_number
* Convert $command\_id to an unsigned integer
* Verify we have the entire message based on $command\_length
* Find $command\_id in switch statement
  + 1, 2, and 9: BIND operations
    - Initialize $smsc\_set (and related variables) for this SMSC set
    - If no SMSCs are connected initiate connections to the SMSCs (MR::prime)
    - If the number of connected SMSCs is lower than the number of available SMSCs, open connections to the rest of the available SMSCs (MR::prime)
    - If no SMSCs connections can be opened, close ESME connection
  + 21: enquire\_link
    - If at least one SMSC is connected return a link\_enquire\_response
    - If no SMSCs are connected do nothing
  + Default
    - No SMSCs: Reply with an error and drop message
    - 1+ SMSCs: Base64 encode the message and call GENERICMESSAGE::create
    - Remove the current message from $incoming\_buf

###### SERVER\_CONNECTED

* Initialize variables required for the SMSC connection
* Initialize $smsc\_set (and related variables) for this SMSC set
* Send SMPP BIND
* Set $peer\_state to ‘waiting\_for\_bind\_resp’
* Begin collecting TCP

###### SERVER\_DATA

* Verify at least 16 bytes of data collected
* Extract $command\_length, $command\_id, $command\_status, $sequence\_number
* Convert $command\_id to an unsigned integer
* Verify we have the entire message based on $command\_length
* Determine if this is a request or a response message
* Find $command\_id in switch statement
  + 1, 2, and 9: BIND request – SMSC should never send these
  + BIND\_RESPONSE
    - If error, log and close connection
    - Set $peer\_state to ‘bound’
    - Add SMSC to $smsc\_list and increment $smsc\_count
    - Use the ‘after’ command to perform the following every 10 seconds
      * Compare the number of connected SMSCs to the number of available SMSCs
      * Open connections to any SMSCs that are available but not connected
    - Forward any queued messages
  + Default
    - ESMEs <= 0: Respond with error and drop message
    - ESMEs >= 1: Base64 encode message and call GENERICMESSAGE::create
    - Remove current message from $incoming\_buf

###### GENERICMESSAGE\_INGRESS

###### MR\_INGRESS

###### MR\_EGRESS

###### GENERICMESSAGE\_EGRESS

###### MR\_FAILED

* Clientside
  + $retry\_count >= max retries
    - Send response to originator
  + $retry\_count < max retries
    - Set nexthop to ‘none’
    - Retry message
* Serverside
  + Log message routing failure
  + Drop message (implicit)

###### CLIENT\_CLOSE

* Delete peer (ESME) entry from $esme\_list table
* Update $esme\_count with number of ESME connections
* If ESME count drops to 0, unbind all SMSCs

###### SERVER\_CLOSE

* Delete peer (SMSC) entry from $smsc\_list table
* Update $smsc\_count with number of ESME connections
* If SMSC count drops to 0, unbind all ESMEs

1. Technically, the forwarding of SMPP traffic to discreet internal virtual-servers is unnecessary and the deployment could be simplified by having the director virtuals run the *smpp\_data\_plane* and *smpp\_mgmt\_plane* iRules directly. However, eliminating the internal (SMSC-set specific) virtuals complicated troubleshooting significantly. Using per-SMSC-set virtuals allows far easier packet captures for traffic related in individual SMSC-sets. [↑](#footnote-ref-2)