SCCP

Table of Contents

The Signaling Connection Control Part (SCCP) is defined in ITU-T Recommendations Q.711-Q.716. SCCP sits on top of Message Transfer Part 3 (MTP3) in the SS7 protocol stack. The SCCP provides additional network layer functions to provide transfer of noncircuit-related (NCR) signaling information, application management procedures and alternative, more flexible methods of routing.

Routing Management

SCCP provides a routing function that allows signaling messages to be routed to a signaling point based on dialed digits, for example. This capability is known as Global Title Translation (GTT), which translates what is known as a global title (for example, dialed digits for a toll free number) into a signaling point code and a subsystem number so that it can be processed at the correct application.

GTT Configuration

GTT is performed in two stages. First is matching the rule and second is actual translation.

For matching the rule, the called party address global title digits are matched with <digits> configured while defining the Rule. Once the digits match actual translation is done. Also for rule matching GT Indicator, translation type, numbering plan and nature-of-address of a rule and of message called party address must be equal.

Matching rule

As explained previously, the <digits> can be divided into sections using the "/" separate character. Each section defines set of digits to be matched. Wild card * can be used to match any digits and ? can be used to match exatcly one digit. For example Rule is to match starting 4 digits (should be 1234) and doesn’t care for rest; the <digits> in the command will be 1234/*. If the Rule is such that starting 3 digits should be 123, doesn’t care for other three digits but last two digits should be 78; the <digits> in the command will be 123/???/78. If digit to digit matching is needed the the <digits> in the command will be exact digits to be matched without sections.

Translation

Resulting Called Party Address will contain:

- PC in the generated CalledPartyAddress is taken from primary/backup address. This PC will not be encoded into a message if the following option is set: sccp set removespc true.
- SSN will be taken from primary/backup address. If it is absent there - will be taken from an original message address. If also absent - SSN will not be included.
- GT type (0001, 0010, 0011, 0100) is taken from from AI (AddressIndicator) of primary/backup address. If AI of primary/backup address contains "no GT" (or bad value) but after address translating we need GT - it will be taken from AI of an original message address.
- NAI, TT, NP (and Encoding schema) is taken from GT of primary/backup address. If no GT in primary/backup address but after address translating we need GT - it will be taken from GT

of an original message address.

- Digits for CalledPartyAddress will be generated in following way. For translation each section in <mask> defined while creating the Rule, defines how replacement operation is performed. If <mask> defines K (KEEP), the originally dialed digits are kept and if <mask> defines R (REPLACE) the digits from primary address or back address are used. The primary/backup address should always define the point code and the translated address will always have this point code. If the primary/backup address defines the subsystem number the translated address will also have this subsystem number. The address-indicator of translated address is always from primary/backup address. See below examples

  1. Example 1 : Remove the Global Title and add PC and SSN

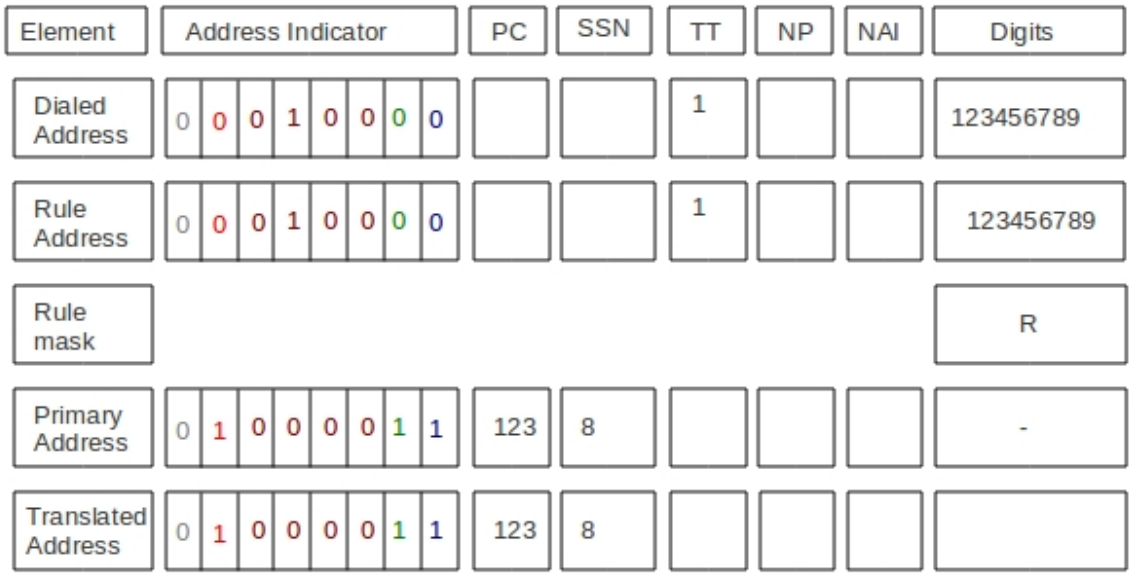

Figure 1. GTT - Example 1

  2. Example 2 : Partial Match Match a eight digit number starting "800", followed by any four digits, then "9". If the translated digits is not null and if the primary/backup address has no Global Title, the Global Title from dialed address is kept with new translated digits.

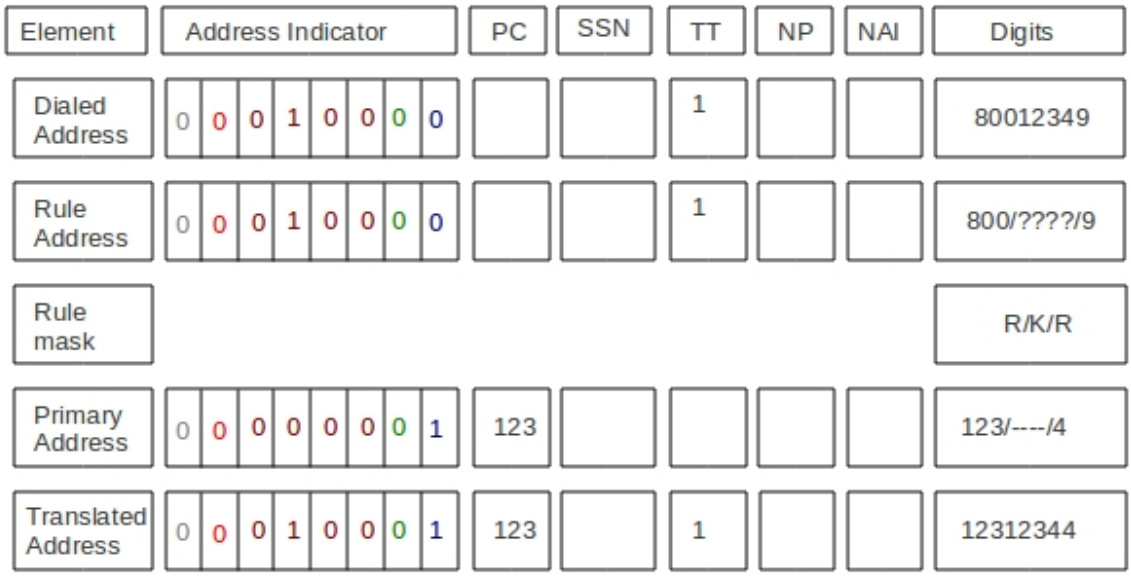

Figure 2. GTT - Example 2

3. Example 3 : Partial Match Match "800800", followed by any digits Remove the first six digits. Keep any following digits in the Input. Add a PC(123) and SSN (8).

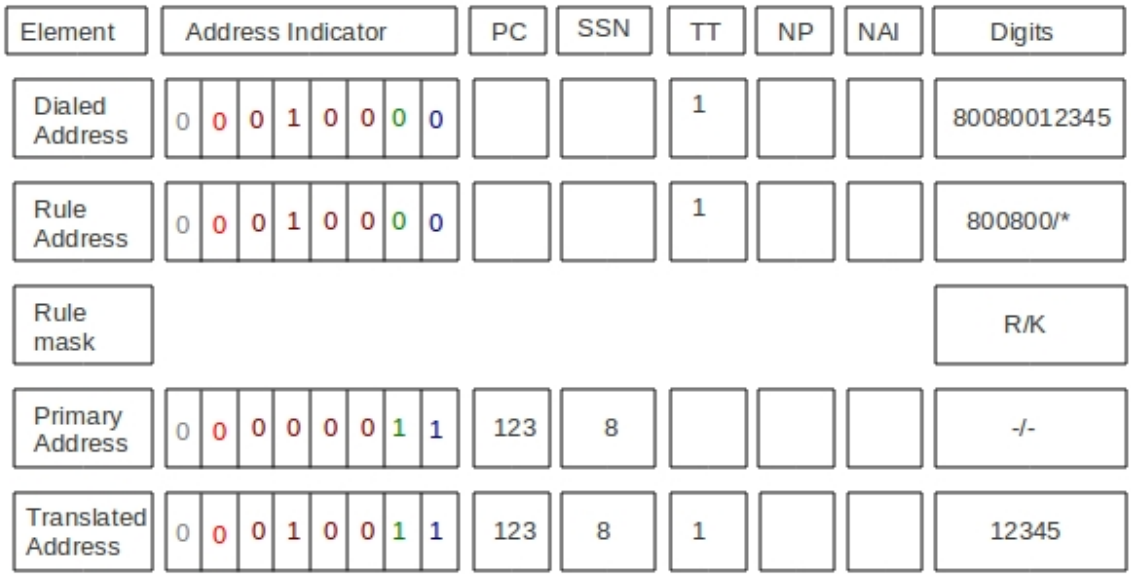

Figure 3. GTT - Example 3

4. Example 4 : Partial Match Match any digits keep the digits in the and add a PC(123) and SSN (8). If the translated digits is not null and if the primary/backup address has no Global Title, the Global Title from dialed address is kept with new translated digits.

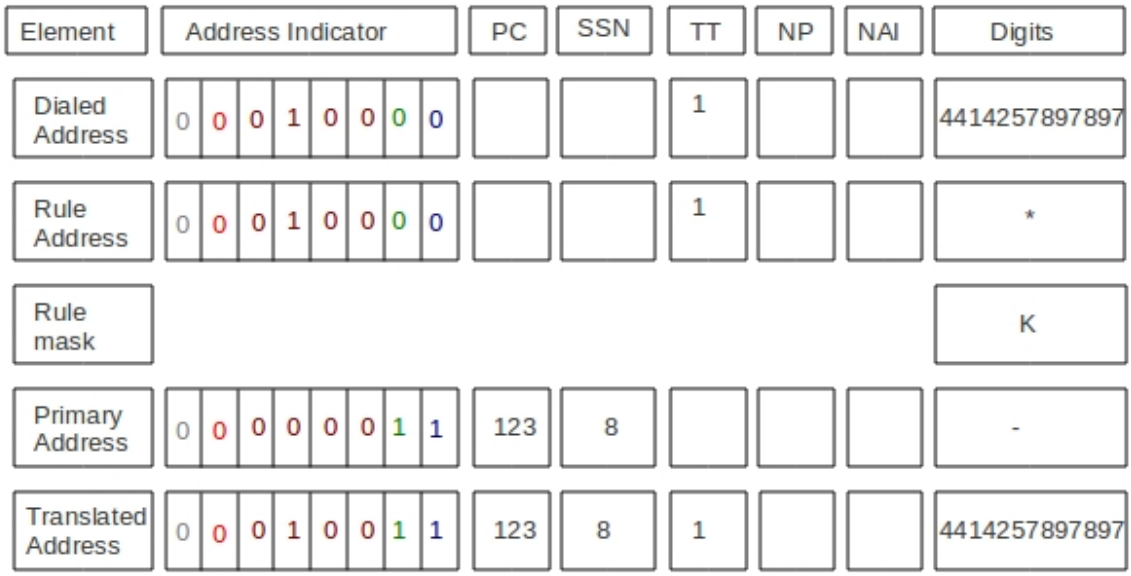

Figure 4. GTT - Example 4

SCCP Usage

The instance of `org.restcomm.protocols.ss7.sccp.SccpStack` acts as starting point. All the sccp messages sent by SCCP User Part are routed as per the rule configured in Router

The term SCCP User Part refers to the applications that use SCCP's services.

The SCCP User Part gets handle to `SccpStack` by doing a JNDI look-up.

`SccpStack` exposes `org.restcomm.protocols.ss7.sccp.SccpProvider` that interacts directly with SccpStack. This interface defines the methods that will be used by SCCP User Part to send connectionless `org.restcomm.protocols.ss7.sccp.message.SccpMessage`, create new connections and register [class]`org.restcomm.protocols.ss7.sccp.SccpListener's to listen for incoming SCCP messages and connection events.

SCCP User Part registers SccpListener for specific local subsystem number. For every incoming SccpMessage or connection event, if the called subsystem matches with this local subsystem, the corresponding SccpListner is called.

`SccpProvider` exposes `SccpConnection` which allows to perform common connection actions such as establishing, resetting, confirming/refusing and disconnecting SCCP connections.

SccpProvider also exposes `org.restcomm.protocols.ss7.sccp.message.MessageFactory` and `org.restcomm.protocols.ss7.sccp.parameter.ParameterFactory` to create new SccpMessage. For transfer data via connectionless service `org.restcomm.protocols.ss7.sccp.message.SccpDataMessage` is used. (This class use UDT, XUDT, LUDT SCCP message type for message transfer.)

SCCP User Part Example #1 (connectionless protocol class)

Below is SCCP User Part example listening for incoming connectionless SCCP message and sending back new message

```
package org.restcomm.protocols.ss7.sccp.impl;

import java.io.IOException;

import org.restcomm.protocols.ss7.indicator.NatureOfAddress;
import org.restcomm.protocols.ss7.indicator.NumberingPlan;
import org.restcomm.protocols.ss7.indicator.RoutingIndicator;
import org.restcomm.protocols.ss7.sccp.RemoteSccpStatus;
import org.restcomm.protocols.ss7.sccp.SccpListener;
import org.restcomm.protocols.ss7.sccp.SccpProvider;
import org.restcomm.protocols.ss7.sccp.SignallingPointStatus;
import org.restcomm.protocols.ss7.sccp.message.SccpDataMessage;
import org.restcomm.protocols.ss7.sccp.message.SccpNoticeMessage;
import org.restcomm.protocols.ss7.sccp.parameter.GlobalTitle;
import org.restcomm.protocols.ss7.sccp.parameter.HopCounter;
import org.restcomm.protocols.ss7.sccp.parameter.SccpAddress;

public class Test implements SccpListener {
        private SccpProvider sccpProvider;
        private SccpAddress localAddress;
        private int localSsn = 8;

        private static SccpProvider getSccpProvider() {
                Mtp3UserPartImpl mtp3UserPart1 = null;
                // ......
                // ......
                SccpStackImpl sccpStack1 = new SccpStackImpl("testSccpStack");
                sccpStack1.setMtp3UserPart(1, mtp3UserPart1);
                sccpStack1.start();
                return sccpStack1.getSccpProvider();
        }

        public void start() throws Exception {
                this.sccpProvider = getSccpProvider();
                int translationType = 0;
                GlobalTitle gt = GlobalTitle.getInstance(translationType,
                                NumberingPlan.ISDN_MOBILE, NatureOfAddress.NATIONAL,
"1234");
                localAddress = new SccpAddress(RoutingIndicator
.ROUTING_BASED_ON_GLOBAL_TITLE, -1, gt, 0);
                this.sccpProvider.registerSccpListener(this.localSsn, this);
        }
```

```
        public void stop() {
                this.sccpProvider.deregisterSccpListener(this.localSsn);
        }

        @Override
        public void onMessage(SccpDataMessage message) {
                localAddress = message.getCalledPartyAddress();
                SccpAddress remoteAddress = message.getCallingPartyAddress();
                // now decode content
                byte[] data = message.getData();
                // processing a request
                byte[] answerData = new byte[10];
                // put custom executing code here and fill answerData
                HopCounter hc = this.sccpProvider.getParameterFactory
().createHopCounter(5);
                SccpDataMessage sccpAnswer = this.sccpProvider.getMessageFactory
().createDataMessageClass1(
                                remoteAddress, localAddress, answerData, message
.getSls(),
                                localSsn, false, hc, null);
                try {
                        this.sccpProvider.send(sccpAnswer);
                } catch (IOException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }
        }

        @Override
        public void onNotice(SccpNoticeMessage message) {
        }

        public void onCoordRequest(int dpc, int ssn, int multiplicityIndicator) {
        }

        public void onCoordResponse(int dpc, int ssn, int multiplicityIndicator) {
        }

        public void onState(int dpc, int ssn, boolean inService, int
multiplicityIndicator) {
        }

        @Override
        public void onPcState(int dpc, SignallingPointStatus status, Integer
restrictedImportanceLevel,
                        RemoteSccpStatus remoteSccpStatus) {
        }

        @Override
        public void onNetworkIdState(int networkId, NetworkIdState networkIdState) {
        }
```

```
        @Override
        public void onConnectIndication(SccpConnection conn, SccpAddress
calledAddress,
                        SccpAddress callingAddress, ProtocolClass clazz, Credit
credit, byte[] data,
                        Importance importance) throws Exception {
        }

        @Override
        public void onConnectConfirm(SccpConnection conn, byte[] data) {
        }

        @Override
        public void onDisconnectIndication(SccpConnection conn, ReleaseCause reason,
byte[] data) {
        }

        @Override
        public void onDisconnectIndication(SccpConnection conn, RefusalCause reason,
byte[] data) {
        }

        @Override
        public void onDisconnectIndication(SccpConnection conn, ErrorCause errorCause)
{
        }

        @Override
        public void onResetIndication(SccpConnection conn, ResetCause reason) {
        }

        @Override
        public void onResetConfirm(SccpConnection conn) {
        }

        @Override
        public void onData(SccpConnection conn, byte[] data) {
        }

        @Override
        public void onDisconnectConfirm(SccpConnection conn) {
        }
}
```

SCCP User Part Example #2 (connection-oriented protocol class)

Below is SCCP User Part example listening for incoming connection-oriented SCCP data message and sending back new data message via already established protocol class 2 connection

```
package org.restcomm.protocols.ss7.sccp.impl;

import org.restcomm.protocols.ss7.indicator.RoutingIndicator;
import org.restcomm.protocols.ss7.sccp.NetworkIdState;
import org.restcomm.protocols.ss7.sccp.RemoteSccpStatus;
import org.restcomm.protocols.ss7.sccp.SccpConnection;
import org.restcomm.protocols.ss7.sccp.SccpListener;
import org.restcomm.protocols.ss7.sccp.SccpProvider;
import org.restcomm.protocols.ss7.sccp.SignallingPointStatus;
import org.restcomm.protocols.ss7.sccp.impl.parameter.ImportanceImpl;
import org.restcomm.protocols.ss7.sccp.impl.parameter.LocalReferenceImpl;
import org.restcomm.protocols.ss7.sccp.impl.parameter.ProtocolClassImpl;
import org.restcomm.protocols.ss7.sccp.impl.parameter.ReleaseCauseImpl;
import org.restcomm.protocols.ss7.sccp.message.SccpConnCrMessage;
import org.restcomm.protocols.ss7.sccp.message.SccpDataMessage;
import org.restcomm.protocols.ss7.sccp.message.SccpNoticeMessage;
import org.restcomm.protocols.ss7.sccp.parameter.Credit;
import org.restcomm.protocols.ss7.sccp.parameter.ErrorCause;
import org.restcomm.protocols.ss7.sccp.parameter.Importance;
import org.restcomm.protocols.ss7.sccp.parameter.ProtocolClass;
import org.restcomm.protocols.ss7.sccp.parameter.RefusalCause;
import org.restcomm.protocols.ss7.sccp.parameter.ReleaseCause;
import org.restcomm.protocols.ss7.sccp.parameter.ReleaseCauseValue;
import org.restcomm.protocols.ss7.sccp.parameter.ResetCause;
import org.restcomm.protocols.ss7.sccp.parameter.SccpAddress;
import org.restcomm.protocols.ss7.scheduler.Clock;
import org.restcomm.protocols.ss7.scheduler.DefaultClock;
import org.restcomm.protocols.ss7.scheduler.Scheduler;

public class Test2 implements SccpListener {
    private SccpProvider sccpProvider;
    private SccpConnection conn;
    private int localSsn = 8;

    private static SccpProvider getSccpProvider() {
        Clock clock = new DefaultClock();
        Scheduler scheduler = new Scheduler();
        scheduler.setClock(clock);

        Mtp3UserPartImpl mtp3UserPart1 = null;
        SccpStackImpl sccpStack1 = new SccpStackImpl(scheduler, "testSccpStack");
        sccpStack1.setMtp3UserPart(1, mtp3UserPart1);
```

```
        scheduler.start();
        sccpStack1.start();

        return sccpStack1.getSccpProvider();
    }

    public void start() throws Exception {
        this.sccpProvider = getSccpProvider();
        this.sccpProvider.registerSccpListener(this.localSsn, this);
    }

    public void stop() {
        this.sccpProvider.deregisterSccpListener(this.localSsn);
    }

    public void connect() throws Exception {
        // assumed that local DPC is 1 and remote DPC is 2, will differ in real life
scenario
        SccpAddress localAddress = this.sccpProvider.getParameterFactory
().createSccpAddress(RoutingIndicator.ROUTING_BASED_ON_DPC_AND_SSN, null, 1,
localSsn);
        SccpAddress remoteAddress = this.sccpProvider.getParameterFactory
().createSccpAddress(RoutingIndicator.ROUTING_BASED_ON_DPC_AND_SSN, null, 2,
localSsn);

        SccpConnCrMessage crMsg = this.sccpProvider.getMessageFactory
().createConnectMessageClass2(8, remoteAddress, localAddress, new byte[] {}, new
ImportanceImpl((byte)1));
        crMsg.setSourceLocalReferenceNumber(new LocalReferenceImpl(1));
        crMsg.setProtocolClass(new ProtocolClassImpl(2));

        conn = this.sccpProvider.newConnection(8, new ProtocolClassImpl(2));
        conn.establish(crMsg);
    }

    public void disconnect() throws Exception {
        conn.disconnect(new ReleaseCauseImpl(ReleaseCauseValue.UNQUALIFIED), new
byte[] {});
    }

    @Override
    public void onConnectIndication(SccpConnection conn, SccpAddress calledAddress,
SccpAddress callingAddress, ProtocolClass clazz, Credit credit, byte[] data,
Importance importance) throws Exception {
        this.conn = conn;
        conn.confirm(null, null, new byte[] {});
    }

    @Override
    public void onData(SccpConnection conn, byte[] data) {
        // decoding of content in byte[] data
```

```
        // processing a request
        byte[] answerData = new byte[10];
        // put custom executing code here and fill answerData
        try {
            conn.send(answerData);
        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }

    @Override
    public void onMessage(SccpDataMessage message) {
    }

    @Override
    public void onNotice(SccpNoticeMessage message) {
    }

    @Override
    public void onCoordResponse(int ssn, int multiplicityIndicator) {

    }

    public void onCoordRequest(int dpc, int ssn, int multiplicityIndicator) {
    }

    public void onCoordResponse(int dpc, int ssn, int multiplicityIndicator) {
    }

    public void onState(int dpc, int ssn, boolean inService, int
multiplicityIndicator) {
    }

    @Override
    public void onPcState(int dpc, SignallingPointStatus status, Integer
restrictedImportanceLevel,
                          RemoteSccpStatus remoteSccpStatus) {
    }

    @Override
    public void onNetworkIdState(int networkId, NetworkIdState networkIdState) {
    }

    @Override
    public void onConnectConfirm(SccpConnection conn, byte[] data) {
    }

    @Override
    public void onDisconnectIndication(SccpConnection conn, ReleaseCause reason,
byte[] data) {
```

```
    }

    @Override
    public void onDisconnectIndication(SccpConnection conn, RefusalCause reason,
byte[] data) {
    }

    @Override
    public void onDisconnectIndication(SccpConnection conn, ErrorCause errorCause) {
    }

    @Override
    public void onResetIndication(SccpConnection conn, ResetCause reason) {
    }

    @Override
    public void onResetConfirm(SccpConnection conn) {
    }

    @Override
    public void onDisconnectConfirm(SccpConnection conn) {
    }
}
```