Introduction

This README describes various timeouts and locks that we can encounter while running an NSO instance. The different conditions will be described and we shall explain how to recognize the various conditions and that the remedy is.

Timeouts

Plain data provider timeout.

A 'config false' data provider is responsible for returning operational data when queried by NSO. If it fails to do so within the stipulated timeout, NSO will close it's end of the socket to the data provider.

An example of such a data provider can be found in packages/dp/src/yang/dp.yang which has:

```
container test-stats {
  config false;
  tailf:callpoint "test-stats-cp";
  list item {
    key name;
    leaf name {
      type string;
    }
    leaf i {
      type int32;
    }
  }
}
```

This data model is served by the code in:

```
packages/dp/src/java/src/com/example/dp/dpDp.java
```

which has a getElem callback that looks like:

```
@DataCallback(callPoint="test-stats-cp",
              callType=DataCBType.GET_ELEM)
public ConfValue getElem(DpTrans trans, ConfObject[] keyPath)
    throws DpCallbackException {

    ConfKey k = (ConfKey)keyPath[1];
    if (k.elementAt(0).toString().equals("k1")) {
        try {
            System.out.println("Sleeping 6 secs");
            Thread.sleep(6000);
        }
        catch (Exception e) {
        }
    }
    return new ConfInt32(44);
}
```

Normally this is no problem, but if we modify ncs.conf so that it has:

```
<japi>
```

```
  <query-timeout>PT4S</query-timeout>
</japi>
```

The /ncs-config/japi/query-timeout will trigger when we execute:

```
$ ncs_cmd -c 'mget /test-stats/item{k1}/i'
FAILED: maapi_get_elem(ms, mtid, &val, argv[0]), Error: external error (19):
application timeout, in function do_maapi_get, line 1449
```

Or, if CLI is used:

```
klacke@ncs> show status test-stats item k1
-----------------------------------------^
syntax error: unknown argument
[error][2016-08-16 15:22:04]

System message at 2016-08-16 15:22:04...
    Subsystem stopped: ncs-dp-1-dp:DpSkeleton
klacke@ncs> klacke@ncs>
System message at 2016-08-16 15:22:06...
    Subsystem started: ncs-dp-3-dp:DpSkeleton
klacke@ncs>
```

Some fairly good messages occur in the various logs. If we start with the NSO main log - ./logs/ncs.log

We see:

```
<CRIT> 16-Aug-2016::15:23:55.456 CEC-cwikstro-mac ncs[31548]: - Daemon ncs-dp-3-
dp:DpSkeleton timed out
```

Which is pretty self explanatory, also the devel.log has a good entry:

```
<DEBUG> 16-Aug-2016::15:26:28.958 CEC-cwikstro-mac ncs[31548]: devel-c get_elem
request for callpoint 'test-stats-cp' path /dp:test-stats/item{k1}/i
<ERR> 16-Aug-2016::15:26:33.971 CEC-cwikstro-mac ncs[31548]: devel-c Worker socket
query timed out daemon 'ncs-dp-3-dp:DpSkeleton' id 25
<ERR> 16-Aug-2016::15:26:33.972 CEC-cwikstro-mac ncs[31548]: devel-c get_elem
error {external_timeout, ""} for callpoint 'test-stats-cp' path /dp:test-
stats/item{k1}/i
<DEBUG> 16-Aug-2016::15:26:36.439 CEC-cwikstro-mac ncs[31548]: devel-c New daemon
connected (name: ncs-dp-4-dp:DpSkeleton, daemon id: 26)
```

Two messages indicating the timeout, also a subsequent message indicating that the recovery code on the Java side is working, the callpoint is re-registered.

The following sequence of events happened here:

1. A user session is created by ncs_cmd
2. A transaction is created by ncs_cmd asking for /test-stats/item{k1}/i
3. NSO identifies the correct handler for the callpoint, and:
4. NSO invokes INIT in that callpoint handler (data provider)
5. NSO invokes GET_ELEM in that data provider.
6. The timeout is triggered, the socket to the data provider is closed by NSO
7. The Java library code sees that a socket to a data provider is closed, the package for that data provider is unloaded

and re-loaded
8. Triggering the registration code to reconnect for the data provider.
9. We're good to go again.

On the Java side, the log says:

```
<ERROR> 16-Aug-2016::15:26:34.472 Dp NCS-DpMUX-3-dp:DpSkeleton: -
com.tailf.conf.ConfException: unexpected end of file
        at com.tailf.conf.ConfInternal.readFill(ConfInternal.java:415)
        at com.tailf.conf.ConfInternal.termRead(ConfInternal.java:185)
        at com.tailf.conf.ConfInternal.termRead(ConfInternal.java:113)
        at com.tailf.dp.Dp.read(Dp.java:1728)
        at com.tailf.ncs.ctrl.NcsDpMux.run(NcsDpMux.java:175)
        at java.lang.Thread.run(Thread.java:745)
Caused by: java.io.EOFException
        ... 6 more
<WARN> 16-Aug-2016::15:26:34.472 NcsDpMux NCS-DpMUX-3-dp:DpSkeleton: - Error in
NcsDpMux
com.tailf.conf.ConfException: unexpected end of file
        at com.tailf.conf.ConfInternal.readFill(ConfInternal.java:415)
        at com.tailf.conf.ConfInternal.termRead(ConfInternal.java:185)
        at com.tailf.conf.ConfInternal.termRead(ConfInternal.java:113)
        at com.tailf.dp.Dp.read(Dp.java:1728)
        at com.tailf.ncs.ctrl.NcsDpMux.run(NcsDpMux.java:175)
        at java.lang.Thread.run(Thread.java:745)
Caused by: java.io.EOFException
        ... 6 more
<ERROR> 16-Aug-2016::15:26:34.473 NcsMain NCS-DpMUX-3-dp:DpSkeleton: - Received
exception from package 'dp' Message: 'unexpected end of file'
<WARN> 16-Aug-2016::15:26:34.473 NcsMain NCS-DpMUX-3-dp:DpSkeleton: - Initializing
redeploy package dp
<INFO> 16-Aug-2016::15:26:36.434 NcsMain NCS-package-restarter: - REDEPLOY ["dp"]
-->
<INFO> 16-Aug-2016::15:26:36.434 NcsMain NCS-package-restarter: - REDEPLOY PACKAGE
COLLECTION  -->
<INFO> 16-Aug-2016::15:26:36.434 NcsMain NCS-package-restarter: - packages
[PackageData=[PackageName="dp", : [Component=
[ComponentName:"DpSkeleton":,ComponentType:"callback",Classes=
[[com.example.dp.dpDp]]]] jars : [file:///Users/klacke/tailf-
src/tailf/examples.ncs/getting-started/developing-with-ncs/19-locks-and-
timeouts/state/packages-in-use/1/dp/private-jar/dp.jar]]]
<INFO> 16-Aug-2016::15:26:36.446 NcsMain NCS-package-restarter: - REDEPLOY PACKAGE
COLLECTION  --> OK
<INFO> 16-Aug-2016::15:26:36.446 NcsMain NCS-package-restarter: - REDEPLOY ["dp"]
--> DONE
```

An interesting variation on this theme is when we repeatedly invoke:

```
$ ncs_cmd -c 'mget /test-stats/item{k1}/i'
```

If we do it sufficiently many times within a short period of time, the error is escalated and we see in ncs.log

```
<CRIT> 16-Aug-2016::15:24:36.427 CEC-cwikstro-mac ncs[31548]: - Daemon ncs-dp-2-
slowsrv:RFSSkeleton died
<INFO> 16-Aug-2016::15:24:36.428 CEC-cwikstro-mac ncs[31548]: - The NCS Java VM
terminated
<INFO> 16-Aug-2016::15:24:36.429 CEC-cwikstro-mac ncs[31548]: - Starting the NCS
Java VM
```

I.e the library code on the Java side decides to terminate entirely, and it is subsequently restarted by NSO - as per the configuration in:

```
klacke@ncs> show configuration java-vm | details
auto-start                     true;
auto-restart                   true;
......
```

The possible remedies to this timeout are:

- Ensure that the data provider code executes faster.
- Raise the ncs.conf timeout for query-timeout.
- The downside of raising the timeout is that it then applies to all data providers. If we know that we have callbacks that are slow, it's possible for the data provider code to ask for more time using the API call DpTrans.dataSetTimeout() This entire discussion also applies to action invocations. It is probably more common with long running action invocations then it is with long running data callbacks.

Fastmap Service create() timeout.

An Fastmap service is defined by it's Yang code and usually (not necessarily) by some mapping code written in Java/Python/Erlang The mapping code gets invoked in a create() call. This call must terminate within a stipulated timeout as configured by:

```
klacke@ncs> show configuration java-vm service-transaction-timeout
service-transaction-timeout 120;
```

The configuration parameter is poorly named as it also applies to services written in other languages.

An example of a service taking too long time to execute can be found in

```
 packages/slowsrv/src/java/src/com/example/slowsrv/slowsrvRFS.java
```

I've configured the service-transaction-timeout to 4 seconds, and the FastMap create() code sleeps for 6 seconds. Committing from the CLI gives:

```
klacke@ncs% set slowsrv s1 sleep-secs 6
[ok][2016-08-22 13:03:24]

[edit]
klacke@ncs% commit
Aborted: application timeout
[error][2016-08-22 13:03:31]
```

The Java log shows similar printouts as in the data provider timeout section above. We see:

```
<ERROR> 22-Aug-2016::13:04:11.118 NcsMain Did-9-Worker-25: - Received exception
from package 'slowsrv' Message: 'Not able to detach service transaction'
<WARN> 22-Aug-2016::13:04:11.118 NcsMain Did-9-Worker-25: - Initializing redeploy
package slowsrv
<ERROR> 22-Aug-2016::13:04:11.300 Dp NCS-DpMUX-5-slowsrv:RFSSkeleton: -
com.tailf.conf.ConfException: unexpected end of file
        at com.tailf.conf.ConfInternal.readFill(ConfInternal.java:415)
        at com.tailf.conf.ConfInternal.termRead(ConfInternal.java:185)
        at com.tailf.conf.ConfInternal.termRead(ConfInternal.java:113)
        at com.tailf.dp.Dp.read(Dp.java:1728)
        at com.tailf.ncs.ctrl.NcsDpMux.run(NcsDpMux.java:175)
        at java.lang.Thread.run(Thread.java:745)
```

```
Caused by: java.io.EOFException
```

and subsequently we also see:

```
<INFO> 22-Aug-2016::13:04:12.714 NcsMain NCS-package-restarter: - REDEPLOY
["slowsrv"]
```

Thus, the Java library code will redeploy the failing package, thus automatically re-registering the callback.

The devel.log shows

```
<ERR> 22-Aug-2016::13:04:10.795 CEC-cwikstro-mac ncs[85651]: devel-c Worker socket
query timed out daemon 'ncs-dp-5-slowsrv:Slowservice' id 9
<ERR> 22-Aug-2016::13:04:10.796 CEC-cwikstro-mac ncs[85651]: devel-c
service_create error {external_timeout, ""} for callpoint 'slowsrv-servicepoint'
path /slowsrv:slowsrv{s1}
```

and the ncs.log shows:

```
<CRIT> 22-Aug-2016::13:04:10.796 CEC-cwikstro-mac ncs[85651]: - Daemon ncs-dp-5-
slowsrv:RFSSkeleton timed out
```

Similar to the data provider callbacks the remedies to this timeout are:

- Make the create() code execute faster.
- Raise the timeout.
- Ask for additional time, by invoking the API method: ServiceContext.setTimeout()

NED timeouts

There are several troublesome timeouts that can be encountered when we connect and manipulate managed devices. These timeouts are primarily controlled by the settings in:

```
admin@ncs> show configuration devices global-settings | details
connect-timeout              20;
read-timeout                 20;
write-timeout                20;
.....
```

It's also possible to change these settings on a per device basis, or alternatively we can create a device profile as in:

```
admin@ncs> show configuration devices profiles
profile slow-devs {
    connect-timeout 40;
    read-timeout    40;
    write-timeout   40;
}
```

and then subsequently refer to this profile in the device meta data config.

For CLI/generic NEDs these timeout numbers are passed to the NED Java code and that code is responsible for implementing the timeouts accordingly.

We don't want these timeouts to ever trigger during normal operations. If one of the timeouts trigger during normal service provisioning, it will manifest itself as an aborted transaction.

Transient errors

When NSO tries to commit data towards a managed device, a class of errors called transient errors, make NSO re attempt. For example, when a NETCONF based device is managed, NSO (sometimes) attempts to lock the configuration store. If this fails, NSO will attempt to take the lock (by default) 6 times, sleeping 3 seconds between each attempt.

This is not configurable. We can look at this behavior in the examples.ncs/getting-started/developing-with-ncs/4-rfs-service example. Here we log in to one of the emulated routers and grab an exclusive lock.

```
$ ncs-netsim cli ex0

admin connected from 127.0.0.1 using console on CEC-cwikstro-mac.local
admin@ex0>configure exclusive
Entering configuration mode exclusive
Warning: uncommitted changes will be discarded on exit
[ok][2016-09-26 10:12:56]
```

At the NSO layer, we turn on southbound tracing to see what happens.

```
$ ncs_cli -u admin

admin connected from 127.0.0.1 using console on CEC-cwikstro-mac.local
admin@ncs> configure
Entering configuration mode private
[ok][2016-09-26 10:14:08]

[edit]
admin@ncs% set devices global-settings trace pretty
[ok][2016-09-26 10:14:21]

[edit]
admin@ncs% commit
Commit complete.
[ok][2016-09-26 10:14:23]

[edit]

admin@ncs% set devices device ex0 config r:sys syslog server 3.4.5.6
[ok][2016-09-26 10:15:00]

[edit]
admin@ncs% commit
Aborted: RPC error towards ex0: lock_denied: the configuration database is locked
by session 12 admin console (cli from 127.0.0.1) on since 2016-09-26 10:12:41
[error][2016-09-26 10:15:18]
```

If we now take a look at the trace log file, we see 8 consecutive sequences the following NETCONF RPCs (3 second intervals)

```
>>>>out 26-Sep-2016::10:15:03.426 device=ex0 session-id=14
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="1">
    <discard-changes/>
</rpc>

>>>>out 26-Sep-2016::10:15:03.426 device=ex0 session-id=14
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
     message-id="2">
  <lock>
```

```
    <target>
      <candidate/>
    </target>
  </lock>
</rpc>

>>>>out 26-Sep-2016::10:15:03.426 device=ex0 session-id=14
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
     message-id="3">
  <get xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
    <filter>
      <netconf-state xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-monitoring">
        <datastores>
          <datastore>
            <name>running</name>
            <transaction-id xmlns="http://tail-f.com/yang/netconf-monitoring"/>
          </datastore>
        </datastores>
      </netconf-state>
     </filter>
  </get>
</rpc>

<<<<in 26-Sep-2016::10:15:03.432 device=ex0 session-id=14
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
           message-id="1">
  <rpc-error>
    <error-type>application</error-type>
    <error-tag>in-use</error-tag>
    <error-severity>error</error-severity>
    <error-message xml:lang="en">the configuration database is locked by session
12 admin console (cli from 127.0.0.1) on since 2016-09-26 10:12:41</error-message>
    <error-info>
      <session-id>0</session-id>
    </error-info>
  </rpc-error>
</rpc-reply>

<<<<in 26-Sep-2016::10:15:03.436 device=ex0 session-id=14
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
           message-id="2">
  <rpc-error>
    <error-type>protocol</error-type>
    <error-tag>lock-denied</error-tag>
    <error-severity>error</error-severity>
    <error-message xml:lang="en">the configuration database is locked by session
12 admin console (cli from 127.0.0.1) on since 2016-09-26 10:12:41</error-message>
    <error-info>
      <session-id>0</session-id>
    </error-info>
  </rpc-error>
</rpc-reply>

<<<<in 26-Sep-2016::10:15:03.437 device=ex0 session-id=14
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
           message-id="3">
  <data>
    <netconf-state xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-monitoring">
      <datastores>
        <datastore>
          <name>running</name>
          <transaction-id xmlns="http://tail-f.com/yang/netconf-monitoring">1474-
877541-281353</transaction-id>
        </datastore>
      </datastores>
    </netconf-state>
  </data>
</rpc-reply>
```

Locks

Debugging locking issues in NSO is hard. When we in a production system start to get failed service provisioning with the dreaded 'database locked' type of errors it's time to bit the bullet. This guide tries to help with that.

NETCONF lock / configure exclusive

There exists a great deal of different locks in the system. The easiest lock to understand is the lock that is taken by CLI command 'configure exclusive'

If we in one CLI do:

```
admin@ncs> configure exclusive
Entering configuration mode exclusive
Warning: uncommitted changes will be discarded on exit
[ok][2016-08-24 13:19:24]

[edit]
admin@ncs%
```

And then subsequently in another CLI session do:

```
admin@ncs> configure exclusive
Error: configuration database locked by:
admin console (cli from 127.0.0.1) on since 2016-08-24 13:19:21
        exclusive
Aborted: configuration locked
[error][2016-08-24 13:19:48]
```

it shows that we cannot take two locks, furthermore

```
admin@ncs> configure
Entering configuration mode private
Current configuration users:
admin console (cli from 127.0.0.1) on since 2016-08-24 13:19:21 exclusive mode
[ok][2016-08-24 13:29:14]

[edit]
admin@ncs% delete aaa
[ok][2016-08-24 13:29:19]

[edit]
admin@ncs% commit
Aborted: the configuration database is locked by session 30 admin console (cli
from 127.0.0.1) on since 2016-08-24 13:19:21
[error][2016-08-24 13:29:35]
```

shows that as long as the exclusive lock is in place, no one can edit the configuration. So who is that pesky 'session 30'

```
admin@ncs% run show users
SID USER  CTX FROM      PROTO   LOGIN    CONFIG MODE
*31 admin cli 127.0.0.1 console 14:04:25 private
30 admin cli 127.0.0.1 console 14:01:00 exclusive
```

Which accidentally also makes the point of not having a shared 'admin' user.

This last operation took 15 seconds to execute:

```
admin@ncs% timecmd commit
Aborted: the configuration database is locked by session 36 admin console (cli
from 127.0.0.1) on since 2016-08-24 13:19:21
Command executed in 15.08 sec

[error][2016-08-24 13:38:48]
```

This is controlled by the ncs.conf parameter:

```
   /ncs-config/commit-retry-timeout (xs:duration | infinity) [PT15S]
       Commit timeout in the NCS backplane. This timeout controls for how
       long the commit operation will attempt to complete the operation
       when some other entity is locking the database, e.g. some other
       commit is in progress or some managed object is locking the
       database.
```

The default value is 15 seconds. For many applications this is way to low. More on this later in this guide.

Similarly, if a NETCONF client executes the RPC, we get similar behavior:

To trigger this, in one shell we do:

```
$ ssh -s admin@localhost -p 2022 netconf
<?xml version="1.0" encoding="UTF-8"?>
    <hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
      <capabilities>
        <capability>urn:ietf:params:netconf:base:1.0</capability>
      </capabilities>
    </hello>
]]>]]>
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
     message-id="1">
  <lock>
    <target>
      <running/>
    </target>
  </lock>
</rpc>
]]>]]>
```

Leaving the SSH connection open, thus holding the lock.

In another we do:

```
$ ncs_cli -u admin

admin connected from 127.0.0.1 using console on CEC-cwikstro-mac.local
admin@ncs> configure
Entering configuration mode private
[ok][2016-08-31 14:44:42]

[edit]
admin@ncs% delete aaa
[ok][2016-08-31 14:44:53]

[edit]

admin@ncs% timecmd commit
```

```
Aborted: the configuration database is locked by session 30 admin ssh (netconf
from 127.0.0.1) on since 2016-08-31 14:42:24
Command executed in 15.06 sec

[error][2016-08-31 14:45:15]

[edit]
admin@ncs% run show users
SID USER  CTX     FROM      PROTO   LOGIN    CONFIG MODE
*31 admin cli     127.0.0.1 console 14:44:39 private
 30 admin netconf 127.0.0.1 ssh     14:42:45 exclusive
[ok][2016-08-31 14:45:31]


admin@ncs% run request system logout user 30
[ok][2016-08-31 14:49:31]

[edit]
admin@ncs% commit
Commit complete.
[ok][2016-08-31 14:49:39]
```

Slow commit that holds lock for too long

Some transactions may run for too long time. Typical scenarios when this happen are:

- We modify remote devices that are slow
- We make massive changes to a device, which then is slow
- The transaction spans a large number of devices.

Regardless, once a transaction has reached a certain point in it's life (we call it trans-lock) no other transactions are allowed to pass that point. Thus at a certain point in the life of a transaction (the trans-lock point) NSO serializes the execution. Only one transaction is allowed be in this state at a given time.

Unless commit-queues are used, the actual modification of managed devices under /devices/device is executed under the trans-lock. Thus, unless commit-queues are used, only one transaction at a time can modify the network.

In this example we added some code using the external DB interface making it possible to have the commit of a transaction take an inordinate amount of time so that we can show the effects of "slow" transactions.

This code resides in packages/extern-db and it has YANG code:

```
container extern {
  tailf:callpoint workPoint;
    leaf sleep-secs {
      type int32;
    }
}
```

and accompanying Java code that sleeps 'sleep-secs' in the commit callback. Thus we have:

```
admin@ncs% set extern sleep-secs 3
[ok][2016-09-12 13:33:55]

[edit]
admin@ncs% timecmd commit
Commit complete.
Command executed in 3.01 sec

[ok][2016-09-12 13:34:00]
```

To drive these slow transactions, we have created an action which creates a user session, a read-write transaction and sets the the /extern/sleep-secs value. This fairly well mimics the use cases cited above with e.g slow devices. This action resides in packages/slowsrv and has Yang code:

```
container example {
  tailf:action a1 {
    tailf:actionpoint a1;
    input {
      leaf sleep-secs {
        type int32;
      }
      leaf system-session {
        type boolean;
      }
    }
  }
}
```

If we invoke that action as:

```
admin@ncs> request example a1 system-session false sleep-secs 2000
```

We get a long running transaction running as the "admin" user. This session shows in CLI under:

```
admin@ncs> show users
SID USER  CTX     FROM      PROTO   LOGIN    CONFIG MODE
*77 admin cli     127.0.0.1 console 14:59:28
 75 admin example 127.0.0.1 tcp     14:52:15 private
 69 admin cli     127.0.0.1 console 14:51:25
```

and is shows from shell command 'ncs --status' as:

```
$ ncs --status
.....
user sessions:

  sessionId=75 2016-09-12 14:52:15 admin@127.0.0.1 example/tcp
com.example.slowsrv.slowTrans.setData
       no locks set
       transaction tid=204 has the trans lock on running
       transactions:
          tid=204 db=running mode=read_write com.example.slowsrv.slowTrans.setData

.....
```

If we now, while this long running transaction resides in the 'trans lock' phase also try to do something else:

```
[edit]
admin@ncs% delete aaa
[ok][2016-09-12 15:01:50]

[edit]
admin@ncs% commit
Aborted: the configuration database is locked by session 75 admin tcp (example
from 127.0.0.1) on since 2016-09-12 14:52:15
   com.example.slowsrv.slowTrans.setData
[error][2016-09-12 15:02:06]
```

What happened here is that this new transaction failed due to the ncs.conf timeout called /ncs-config/commit-retry-timeout The default value for this timeout is 15 seconds. For many deployments this is way too low. It can be argued that the timeout should be 'infinity' In general, we never want northbound transactions to fail due to other long running transactions, we want all transactions to be queued and run eventually. Long running transactions are not an error, it's normal.

In NSO 4.2 and earlier, so called 'system sessions' did not show in the output from 'ncs --status' making it inherently difficult to track down 'database locked' situations. As of 4.3, also 'system sessions' show there and we also get additional data indicating which code (if any) started the session.

The remedy to this is to set the ncs.conf parameter /ncs-config/commit-retry-timeout to infinity or to a very large number such as 15 minutes. I recommend infinity.

Slow service create code

If we have service create() code that is slow, we might encounter the timeout controlled by the poorly named config parameter /java-vm/service-transaction-timeout

That is one problem with slow create() code, another problem is that by default, the create() code runs under the previously discussed 'trans-lock' Thus, at most one transaction at a time can run inside service create().

Common examples of slow services are:

- Code that is afraid, code that executes various checks on the configuration before applying the changes to the network.
- Code that creates large diff sets. This is sometimes common in NFV settings where the entire device configuration is part of the service. If the device configuration is large, it may take some time to apply the configuration.

The remedy to the locking problem for slow services is to declare the service as a PRE_LOCK_CREATE service instead of a CREATE service.

The Java code in slowsrvRFS.java in packages/slowsrv shows two services, one standard CREATE and one PRE_LOCK_CREATE. Both have a leaf:

```
leaf sleep-secs {
  type int32;
}
```

and both read that input leaf and sleeps as:

```
ConfInt32 v = (ConfInt32) service.leaf("sleep-secs").value();
int sleep = v.intValue();
Thread.sleep(sleep * 1000);
```

The service that use PRE_LOCK_CREATE will execute concurrent, thus if we create two services of the PRE_LOCK_CREATE variant simultaneously, they will both execute concurrently, whereas if we simultaneously create two services of the CREATE variant, the two transaction will be serialized.

Hence, PRE_LOCK_CREATE services increase overall NSO system throughput. The only downside to PRE_LOCK_CREATE is that the service creation code has to be thread safe, on that other hand that is probably a good idea anyway.

CDB subscriber that never finishes

A common error over the years, especially for Java based CDB subscribers is code that subscribes to CDB and forgets to call sync() on the subscription socket. A CDB subscriber needs to be an application component. It must run as a thread. This is the same for Java/Erlang/Python subscribers. The archetypical Java CDB subscriber looks something like:

```
public void run() {
    try {
        while (true) {
            int[] points;

            try {
                points = sub.read();
            } catch (Exception e) {
                return;
            }

            EnumSet<DiffIterateFlags> enumSet =
                EnumSet.<DiffIterateFlags>of(
                    DiffIterateFlags.ITER_WANT_PREV,
                    DiffIterateFlags.ITER_WANT_ANCESTOR_DELETE,
                    DiffIterateFlags.ITER_WANT_SCHEMA_ORDER);

            int th = -1;
            try {
                th = maapi.startTrans(Conf.DB_RUNNING, Conf.MODE_READ);
                NavuContainer root = new NavuContainer(maapi, th, Ncs.hash);
                sub.diffIterate(points[0], new Iter(sub, root),
                                enumSet, null);

                sub.sync(CdbSubscriptionSyncType.DONE_OPERATIONAL);
            } catch (Exception e) {
                // ignore
                ;
            }
            finally {
                maapi.finishTrans(th);
            }
        }
    } catch (Exception e) {
        logger.error("", e);
    }
}
```

With the above code, it's paramount that the call to sub.diffIterate() never throws. If it does, we'll throw over the call to sub.sync() and the CDB socket is locked for ever.