IBM DB2® Universal Database™
DB2 Problem Determination Tutorial Series

Connectivity Problem Determination

Table of Contents

Connectivity Problem Determination

About this tutorial

Tutorial objectives

The objective of this tutorial is to teach you how to identify the source of common communication problems in DB2 and to understand the differences involved in diagnosing two- and three-tier environments.

Audience and assumption

This tutorial is intended for people who are already familiar with the use of DB2, and who want to focus on problem determination techniques where communication errors are concerned. The prerequisite is that they should already have a basic knowledge of the setup of a connection between a client and a server, as well as have an understanding of what a multi-tier environment is. They must have a general understanding of the protocols DB2 uses, in particular TCP/IP.

Pre-tutorial setup

Most examples in this tutorial were written for a Microsoft® Windows® operating system, but all concepts apply to all operating systems and the exercises can be completed on any DB2 UDB distributed system.

All of the examples in this tutorial are given using TCP/IP, however the steps to debug in the other supported protocols are quite similar.

In order to work through the examples in this tutorial you will require the following environment:

1. A server, on which you have the following things:
    - Installed DB2 and the DB2 samples
    - An instance with the DB2 SAMPLE database created
    - SYSADM authority on the SAMPLE database
2. A client, on which you have the following things:
    - Installed DB2
    - An instance in which you have cataloged the server's SAMPLE database, via TCP/IP
    - SYSADM authority on the local instance
    - A successful connection to the server's SAMPLE database.

Tutorial conventions used

When a tool or utility is first mentioned it is shown in **bold** text.

All commands statements and their outputs are shown in a `monospace` font.

Some examples show specific command options that might change over time, which will always be documented in DB2 UDB documentation.

About the author

Lisa Cawley, B.C.S is a DB2 Certified Advanced Technical Expert and a senior member of the DB2 UDB Worldwide Support team at the IBM Toronto Software Laboratory. She currently provides technical support to DB2 customers worldwide in the DB2 Tools, Connectivity and Extenders front-line support team.

You can contact Lisa by going to the IBM Global Directory at http://www.ibm.com/contact/employees/us and searching under the Canada Employee's directory.

Understanding your environment

Defining the systems involved

Before beginning to analyze a connectivity problem, it is imperative that you understand your environment. You should know three pieces of information (as a minimum):

1. Operating systems
2. DB2 products, versions and FixPak levels
3. Communication protocols used between each tier

If you are working in a multi-tier model (i.e. the client tier is supplemented by one or more middle tiers) then it often helps to draw a diagram of your environment. Here is an example:

Another term that you might hear is that of "application requester (AR)" and "application server (AS)". These terms come from DRDA (Distributed Relational Database Architecture), which is a set of protocols used in DB2 v7 when communicating with mainframe databases. In DB2 v8, DRDA is used in communication between all DB2 servers, not just when connecting to mainframes. Thus the concept of the AR and AS is one you'll see quite often. The application requester is the code that handles the application side of a distributed connection; it is the machine that you are connecting "from". The application server is the code that handles the database side of the connection; it is the machine that you are connecting "to".

Where does the problem lie?

The goal when analyzing communication problems is to narrow down *where* the problem originates. The reason for focusing on understanding your environment is to eliminate as many layers of complexity as possible. This can be accomplished as follows:

1. Are you encountering the communication error in an application running on the client? If so, try recreating it from the DB2 command line, thus eliminating the application from the picture.
2. Are you working in a three-tier environment, with the error being returned to the client? Try recreating the problem directly from the middle tier (which is often referred to as a "gateway"). This eliminates the client and the network between the client and the gateway from the picture.

It is likewise important to isolate whether the problem you are encountering is limited to DB2. For example, if you are encountering network errors, is it only via DB2 connections that you are encountering network errors? This becomes especially important when the error is intermittent, as there is a greater likelihood that the problem is outside of DB2 in those situations.

Tools to determine where the problem lies

How do you establish whether or not the problem is specific to DB2? There are tools specific to each protocol that can be used to verify that communication between the participating machines is possible. Here are some examples:

TCP/IP:

```
ping <hostname or ip address>
```

Try it out! Your successful result should look something like this:

```
Pinging myserver [1.23.45.678] with 32 bytes of data:
Reply from 1.23.45.678: bytes=32 time=921ms TTL=253
Reply from 1.23.45.678: bytes=32 time=721ms TTL=253
Reply from 1.23.45.678: bytes=32 time=942ms TTL=253
Reply from 1.23.45.678: bytes=32 time=80ms TTL=253

Ping statistics for 1.23.45.678:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milliseconds:
    Minimum = 80ms, Maximum =  942ms, Average =  666ms
```

APPC:
Each different APPC communication product will have its own method of verifying that a successful LU to LU session can be established. For example, on AIX, you could start the SNA subsystem, start the node, start the link station, and then test a session, all via the smit utility. Please refer to your specific APPC communication product's documentation for more details.

Named Pipes:
Try issuing a `net use` command from the client to a shared folder on the server.

The PCT tool

To assist in debugging the communication setup between the client and the server, DB2 provides a tool called the Protocol Communications Tester. The tool is launched by means of the **pct** command, which is found in the sqllib/bin directory. It is a standalone tool designed to verify that a given LAN protocol is functional, and that two endpoints can communicate with each other via that LAN protocol.

The tool can help solve protocol configuration-type problems because it effectively removes DB2 from the picture until communication has been proven successful. If you cannot pass these communication tests that are external to DB2, this indicates that either your error is in the network layer or else you have not used correct protocol-specific values in the tool. DB2 will not be able to connect across this network until the network problems are corrected and the proper protocol-specific values are identified.

For usage information related to this tool, please refer to the readme.pct file located in the same directory as the executable file. The PCT tool can test the following protocols:

- NetBIOS
- TCP/IP
- IPX/SPX

Configuring PCT on the server

We will now step through an example of using the TCP/IP version of PCT. To test connections made over IPX/SPX or NETBIOS protocols, the PCT instructions are very similar. The main difference is that you enter `pctn` and `pcti` where we entered `pctt`.

On your server, do the following:

1. Verify what your server's hostname and IP address are via the following command: `pctt -r`

```
D:\sqllib\bin>pctt -r
TCPIP Resources...
+-------------------------------------------+
| HostName   : myserver                     |
| HostAddress: myserver.ibm.com             |
| IPAddress  : 123.456.7.890                |
+-------------------------------------------+
```

2. Generate a pct.ini file: `pctt /gt`

   The resultant pct.ini file will appear as follows:

```
#-----------------------------------------------------------
# PCT.INI
# Protocol Communication Tester INI File
# This file contains the Protocol Specific Configuration
# Parameters & values to be used in conjunction with
# the Protocol Communication Tester (PCTx).
#
# Defaults will be used if the parameter is not listed.
#-----------------------------------------------------------

[TCPIP]
ServerHostName =
ServerHostAddress =
ServerIPAddress =
ServiceName =
ServerPort = 49433
```


Ensure that you run this command in a directory to which you have write access, or else the pct.ini file will not be created successfully.

3. Edit the pct.ini file to reflect the protocol-specific parameter values for this server and for the particular instance that your client will be trying to connect to. When you use this tool to debug problems in DB2, it is important to configure the pct.ini file with the exact values that your DB2 instance will use to listen for incoming client requests. For example, you should set the ServiceName or ServerPort value in pct.ini to the same value as is found in the SVCENAME field in the output from the command `db2 get database manager configuration`. Note that you only need to enter a value for either the ServerHostName or the ServerIPAddress. The same holds true for the ServiceName, ServerPort pair. Either one alone will suffice.

Starting PCT on the server

Note: The PCT tool must be running on the server prior to launching it on the client, otherwise the PCT tool on the client will fail, indicating that the connection was refused.

When PCT starts successfully on the server, you see output like this:

```
lcawley@steel:/home/lcawley> pctt s
- Reading configuration parameters
==> pct.ini file was found... using file specified protocol values!
 Protocol Tester values...
   Client/Server          : Server
   Connections            : 1
   Buffer Size            : 500
   Transaction Iterations : 1
   Trace                  : OFF
   Service                : Send/Recv/Verify
   Keep Connections       : NO
   Delayed Send (secs)    : 0
 Local TCPIP specific values...
          Hostname        : myserver
          HostAddress     : myserver.ibm.com
          IPAddress       : 123.456.7.890
```

The information picked up from the pct.ini file is shown in the output above. The output continues as follows...

```
- Initializing the Protocol             Date: 10-22-2002  Time: 0:39:5:45
| retcode = <    0>  ----[ TCPIP.socket     ]-----[  SUCCESS  ]------------
| retcode = <    0>  ----[ TCPIP.setsockopt ]-----[  SUCCESS  ]------------
| retcode = <    0>  ----[ TCPIP.bind       ]-----[  SUCCESS  ]------------
| retcode = <    0>  ----[ TCPIP.listen     ]-----[  SUCCESS  ]------------
- Listening for Remote Clients          Date: 10-22-2002  Time: 0:39:5:46
```

At this point the server side of the PCT tool is waiting to receive a communication from the client.

To start the PCT listener, issue the following command: `pctt s`

Configuring PCT on the client

Now you need to set up the PCT tool on the client, in order to complete the test.

1. Generate a pct.ini file: `pctt /gt`. The file will be blank, and appear exactly as it did when performing this step on the server.

2. Edit this pct.ini file to reflect the protocol-specific parameter values for the server to which you want to connect. In order to use this tool to debug problems in DB2, it is important to fill out the pct.ini file with the exact values that you used when catalogging the remote DB2 server from the client. For example, if you used the server's hostname (as opposed to the IP address) when you ran the CATALOG TCPIP NODE command on the client, then do the same in the pct.ini file. Likewise if you used a servicename (as opposed to a port number), then do the same in pct.ini. Here is an example of an updated pct.ini file:

```
#-----------------------------------------------------
# PCT.INI
# Protocol Communication Tester INI File
# This file contains the Protocol Specific Configuration
# Parameters & values to be used in conjunction with
# the Protocol Communication Tester (PCTx).
#
# Defaults will be used if the parameter is not listed.
#-----------------------------------------------------

[TCPIP]
ServerHostName = myserver.ibm.com
ServerHostAddress =
ServerIPAddress =
ServiceName =
ServerPort = 50000
```

3. Start the server listener: `pctt c`

Successful PCT output

By default, PCT will perform an endpoint--to--endpoint connect operation, followed by send and receive operations with data verification, and finally it will disconnect. Successful results appear as follows:

```
C:\temp>pctt c
- Reading configuration parameters
==> pct.ini file was found... using file specified protocol values!
 Protocol Tester values...
   Client/Server          : Client
   Connections            : 1
   Buffer Size            : 500
   Transaction Iterations : 1
   Trace                  : OFF
   Service                : Send/Recv/Verify
   Keep Connections       : NO
   Delayed Send (secs)    : 0
 Local TCPIP specific values...
          Hostname        : myclient
          HostAddress     : myclient.ibm.com
          IPAddress       : 98.765.4.321
```


(The results of a failed connection attempt are shown later.) The information in the above output shows the PCT tool's default settings as well as the protocol-specific information about your client machine.

```
Server TCPIP specific Values...
   Server Hostname        : myhostname.ibm.com
   Server HostAddress     : myhostname.ibm.com
   Server IPAddress       : 123.456.7.890
  Service Name           :
   Server Port            : 50000
```

The information above was picked up from your client's pct.ini file and shows the values that will be used to identify the server.

```
-Initializing the Protocol   Date: 10/22/02  Time:00:45:26:791
-Connecting to Remote System Date: 10/22/02  Time: 00:45:26:801
| retcode = <0>  ----[ TCPIP.socket     ]-----[SUCCESS]--------
| retcode = <0>  ----[ TCPIP.connect    ]-----[SUCCESS]--------
- Connection established! 1  Date: 10/22/02  Time: 00:45:26:861
```

At this point, the connection step of the test has been completed.

```
- Sending Service Request    Date: 10/22/02  Time: 00:45:26:861
| retcode = <0>  ----[ TCPIP.send       ]---[Bytes=  316 ]-----
- Receiving                  Date: 10/22/02  Time: 00:45:26:871
| retcode = <0>  ----[ TCPIP.receive    ]---[Bytes= 4 ]--------
| retcode = <0>  ----[ TCPIP.receive    ]---[Bytes=  312 ]-----
- Server info:
    Computer Name   : MYHOST
    Operating System: AIX
    Version         : 4.3
    Hostname        : myserver
-Send/Receive/Verify Data Loop Date:10/22/02 Time: 00:45:26:971
- Sending Data               Date: 10/22/02  Time: 00:45:26:971
| retcode = <0>  ----[ TCPIP.send       ]---[Bytes=  500 ]-----
- Receiving                  Date: 10/22/02  Time: 00:45:26:971
| retcode = <0>  ----[ TCPIP.receive    ]---[Bytes=    4 ]-----
| retcode = <0>  ----[ TCPIP.receive    ]---[Bytes=  496 ]-----
```

By default, the PCT tool passes some data from the server to the client, to verify the connection. Then the connection is closed (see below).

```
- Terminating the Connection Date: 10/22/02  Time: 00:45:27: 31
| retcode = <0>  ----[ TCPIP.sock_close ]-----[SUCCESS  ]------
+-----------------------------------------------+
|  Protocol Connection Test Completed - SUCCESS  |
+-----------------------------------------------+
```

Unsuccessful PCT output

A failure to complete the test successfully would result in the following error message:


```
- Reading configuration parameters
==> pct.ini file was found... using file specified protocol values!
 Protocol Tester values...
   Client/Server          : Client
   Connections            : 1
   Buffer Size            : 500
   Transaction Iterations : 1
   Trace                  : OFF
   Service                : Send/Recv/Verify
   Keep Connections       : NO
   Delayed Send (secs)    : 0
 Local TCPIP specific values...
          Hostname        : myclient
          HostAddress     : myclient.ibm.com
          IPAddress       : 98.765.4.321

 Server TCPIP specific Values...
   Server Hostname        : myserver
   Server HostAddress     : myserver.ibm.com
   Server IPAddress       : 1.23.45.678
   Service Name           :
   Server Port            : 99999

- Initializing the Protocol    Date: 11/23/02 Time:22:18:12:507
- Connecting to Remote System  Date: 11/23/02 Time:22:18:12:517
| retcode = <    0>  ----[ TCPIP.socket ]-----[SUCCESS ]-------
| retcode = <10061> +----[ TCPIP.connect]-----[ERROR   ]-------
+===== ERROR ==================================================
| retcode = <10061> ->  Connection refused
+==============================================================
```

The return code given here is a TCP/IP return code 10061 which maps to CONNECTION REFUSED. The cause in my example was an incorrect ServerPort value in the client's pct.ini file. It did not match the ServerPort value defined in the server's pct.ini file.

If you receive errors such as this via the PCT tool, do not go any further with problem determination in DB2 until you have rectified these problems that are external to DB2.

Problem determination

A common DB2 communication error

Once you have clarified your environment setup, and have narrowed the error down to as simple a circumstance as possible, and have determined that you can only recreate the problem via DB2...where do you go from there?

Communication errors within DB2 very typically appear as SQL30081 error messages. The error message will contain a protocol-specific return code, which you can look up in the protocol's error definitions. However, quite often the error codes alone do not provide very straightforward indications of what the cause of the problem was.

The most common communication error message in DB2 is this one:

```
SQL30081N A communication error has been detected. Communication protocol being
used: <protocol>. Communication API being used: <interface>. Location where the
error was detected: <location>. Communication function detecting the error:
<function>. Protocol specific error code(s): <rc1>, <rc2>, <rc3>
```

To simulate this error very simply, go to your server and issue a `db2stop` command on the instance that your client connects to. Then go back to your client and try to connect to the server. You will most likely receive the following error on the client:

```
SQL30081N A communication error has been detected.  Communication
protocol being used: "TCP/IP".  Communication API being used: "SOCKETS".
Location where the error was detected: "".  Communication function
detecting the error: "connect".  Protocol specific error code(s):
"10061", "*", "*". SQLSTATE=08001
```

The following error message is also written in the db2diag.log (with the DIAGLEVEL set to 2 or higher in the client's database manager configuration parameters):

```
2002-11-23-22.38.25.935000   Instance:DB2   Node:000
PID:1320(db2bp.exe)   TID:324  Appid:091A8A2E.07DF.021124033326
common_communication  sqlcctcpconnr   Probe:110
DIA3202C The TCP/IP call "connect" returned an errno="10061".
```

As you can see, both of these error messages tell you that a protocol-specific error of 10061 is being returned on a TCP/IP connect operation. This is the same CONNECTION REFUSED error code that we saw in the output from PCT when the client side of the tool used an incorrect port number to try its connection. However, that's not the case here. In this case, the client was using the correct port number, but the instance was not started for listening on that port number on the server. Thus, you can see that protocol-specific error codes alone are not always sufficient to debug the cause of a problem.

Do not forget to start the DB2 instance on your server again (via the `db2start` command) so that you can continue to use this connection successfully in future exercises.

Problem determination techniques

Since we have seen that error messages alone don't always pinpoint the cause of a communication problem, we need to rely on other problem determination techniques. What follows is a list of four questions to help you in your investigation. Asking yourself these questions when you first tackle a communication problem will help you to determine the exact nature of the situation:

1. Is the problem intermittent or persistent?
2. Can you connect locally on the server?
3. Has the connection ever worked?
4. Does it fail for all users or from all clients?

The implications of your responses to these questions are detailed in the following pages.

Question #1: Intermittent vs persistent

Question#1: Is the problem intermittent or persistent?

If the problem is persistent, i.e. it can be consistently recreated with a set of simple steps, and then it is most likely a configuration issue. Your focus should be on confirming the setup on the client and server. (See section 4 for an example of how to verify the setup)

If the problem is intermittent, then it will be a bit more difficult to debug. It is not the error message itself that is different -- usually you will get an SQL30081 error message just like you do for persistent problems. The difficulty lies in the fact that you cannot recreate it at will, and thus have fewer chances of capturing and diagnosing the error. To pursue intermittent error messages, you will need to investigate the circumstances of the error, for example:

- What is the network state at the time of the problem?
- What is the frequency of the occurrence?
- Look for patterns as to what events are occurring at the time.

Depending on how frequently the errors are occurring, traces may be hard to obtain. It is understandable that you may not want to have traces running for extended periods of time just to capture the error, due to the performance impact traces incur -- particularly if the problem is occurring on a production system. Thus you frequently need to rely on the information in the db2diag.log file for these types of problems.

Intermittent errors and idle threads

One thing to keep in mind when you investigate intermittent errors is how long your connection was idle prior to the error occurrence. If you are receiving a communication error after the connection has been sitting idle for some time -- particularly when a mainframe is involved -- you

should check on the server at the network level for time-out values. For example, when you are attempting to connect to DB2 running on z/OS, check the value of your IDLETHREADTIMEOUT on the host. If the thread on the host times out while DB2 UDB is waiting for further information or requests from client, the client may not know that the host is gone until it next tries to use that connection. Likewise if the client ends abnormally in a three-tier environment, the server may not be notified of the lost connection until the next time that it tries to return data to the client.

If your protocol is TCP/IP, you can tweak KEEPALIVE and KEEPALIVEINTERVAL parameters at the network level, so that the loss of a connection is recognized more quickly. Keep in mind that KEEPALIVE settings affect all TCP/IP applications running on the machine. You will find that in most environments, the default KEEPALIVE value is around 2 hours.

Intermittent errors and query time-out settings

Another thing to check when investigating intermittent communication errors is whether the communication error occurs after a long wait for a query to return from the server. If that is the case, the client application may have reached its QUERYTIMEOUT threshold and canceled the query. The QUERYTIMEOUT value is set in your application, and is used commonly in ODBC applications. If the QUERYTIMEOUT value has been reached and the data has not yet been returned, an SQLCancel request is sent to the server.

In a three-tier environment where a mainframe is involved, a query is canceled on the mainframe by terminating the connection from the gateway to the host. Thus a communication error results. To determine whether QUERYTIMEOUT is implicated when you experience communication errors in an application, set QUERYTIMEOUTINTERVAL=0 in the [COMMON] section of the db2cli.ini file on your client. This causes QUERYTIMEOUT requests coming from the application to be ignored. This is not a permanent solution, but is often used as a temporary means of confirming that QUERYTIMEOUT values are indeed the source of the problem. Tweaking the QUERYTIMEOUT value in your application or investigating the cause of the lengthy response time on the query is your best course of action in this case.

Intermittent errors and connection pooling

One last situation to consider when diagnosing intermittent errors is connection pooling. Aside from any connection pooling that you may be doing at the application layer (via WebSphere or ODBC for example), DB2 does its own pooling of connections between DB2 UDB and mainframes. Thus even if you disconnect a connection to the mainframe, it may actually still be maintained in a pool of connections for future use. If it is not used again right away, it may sit idle long enough to run into the IDLETHREADTIMEOUT situation described earlier in Section 3: "Intermittent errors and idle threads". When retrieving a connection from the pool, DB2 will test the connection to ensure that it is still valid, prior to giving that connection to a new application. However, it can be confusing if you look at messages on the mainframe and see idle thread timeout messages about connections that you thought no longer existed.

If you are connecting from a workstation directly to a mainframe, a connection will not be pooled unless you have either (a) set DB2CONNECT_IN_APP_PROCESS (a db2set registry profile setting) to NO; or (b) used a *loopback* connection (We will discuss loopback connections in greater detail in Section 3:"Question #2: Can you connect locally?"). These two actions also affect monitoring when you connect directly to a mainframe. You need to perform one of these if you want to see information about the locally initiated connection in the output from a `db2 list dcs applications` command. Please note that neither of these actions is possible if you use DB2 Connect Personal Edition to access the mainframe directly, as that product does not have these capabilities.

While there is no permanent method of disabling DB2's connection pooling, a temporary method of doing so is to set NUM_POOLAGENTS (a parameter in the database manager configuration settings) to 0. On rare occasions, you might do this while diagnosing a communication error, if you are concerned that connection pooling may be complicating the issue. Note this is not a very viable setting to use permanently because it affects all agents, not just those associated with pooled connections, and thus results in much greater overhead since each and every agent must be created from scratch.

Question #2: Can you connect locally?

This is a very key question, and should be something that you ask yourself every time that you encounter a communication error. If you cannot connect locally at the time of the error, then you can eliminate the client and the communication layer entirely from the picture and focus instead on recreating and diagnosing the error directly on the server.

If a local connection is indeed successful, then the next step is to simulate a remote connection directly on the server. This is done to reduce the layers of complexity by recreating the entire problem on one machine. This is often referred to as a loopback. The loopback is just a method of cataloging the local database as if it were on a remote machine, as described here. Do all of these actions on your server:

1. Look at the current catalog for the sample database by issuing the command `db2 list db directory`:

```
               Database X entry:

 Database alias                 = SAMPLE
 Database name                  = SAMPLE
 Database drive                 = C:\DB2
 Database release level         = 9.00
 Comment                        =
 Directory entry type           = Indirect
 Catalog node number            = 0
```

   When you have finished, there will be another catalog listed here as remote, but ultimately pointing to the same local database. There are methods of performing loopback catalogs such that the loopback alias name matches

what the original database name was, however we will not be doing those steps. Those steps are often followed when an application will be using the loopback catalog permanently, and you do not want to have to change the way the application or script was referring to the database. We are keeping it simple for the sake of this example.

2. To verify the port number or service name that the instance is listening on for incoming requests, issue the command `db2 get dbm cfg`

```
...
NetBIOS Workstation name              (NNAME) =
TCP/IP Service name                   (SVCENAME) = db2cDB2
 APPC Transaction program name        (TPNAME) =
 IPX/SPX File server name             (FILESERVER) =
 IPX/SPX DB2 server object name       (OBJECTNAME) =
 IPX/SPX Socket number                (IPX_SOCKET) = 879E
...
```

This information, along with the hostname or IP address of this server, is all that you need in order to catalog the node. Note that if the client is experiencing the communication error and the catalog uses the IP address rather than the hostname of the server, then we should do the same in our loopback catalog as well for consistency.

Setting up a loopback connection

1. To catalog the loopback node, issue this command:

```
db2 catalog tcpip node loopnode remote <hostname/ip address>
server <service name/port number>
```

The port number or service name used in this command must match the port number or service name seen in the `db2 get dbm cfg` output. You can then see the resultant node catalog by issuing `db2 list node directory`.

```
Node X entry:

 Node name                      = LOOPNODE
 Comment                        =
 Protocol                       = TCPIP
 Hostname                       = myhostname
 Service name                   = db2cDB2
```

2. To catalog the loopback database, issue this command:

```
db2 catalog db <local db name> as <loop back db alias> at
node <loop back node name, per step 1. above>
```


After that, when you execute db2 list db directory, the database directory will show the two related database catalogs as follows:

```
Database X entry:

 Database alias                 = SAMPLE
 Database name                  = SAMPLE
 Database drive                 = C:\DB2
 Database release level         = 9.00
 Comment                        =
 Directory entry type           = Indirect
 Catalog node number            = 0

Database Y entry:

 Database alias                 = LOOPDB
 Database name                  = SAMPLE
 Node name                      = LOOPNODE
 Database release level         = 9.00
 Comment                        =
 Directory entry type           = Remote
 Catalog node number            = -1
```

Testing a loop-back connection

To connect to the loopback database, enter `db2 connect to <loop back db alias> user <userid>`. You will be prompted for a password.

```
C:\PROGRA~1\SQLLIB\BIN>db2 connect to loopdb user tester

   Database Connection Information

 Database server        = DB2/NT 7.2.5
 SQL authorization ID   = TESTER
 Local database alias   = LOOPDB
```

If the connection works via this loopback connection test, but failed from a remote client, then you might want to investigate these areas:

A. Is there a firewall between the client and the server?
B. Do the node and database catalog on the client match those we performed here on the server? Note that the service name (if used) could be different in the client's node catalog compared to this one, but that the port number that it maps to on each machine should nonetheless match.

If the connection fails via the local loopback test, then you can ignore the client entirely from this point on. If the error is persistent, it is likely a configuration issue, and the server is not correctly set up to listen for incoming requests. You would thus need to focus on confirming the server configuration values.

Question #3: Has this connection ever worked?

If it is a first-time setup, there is a strong likelihood that this is a configuration issue. First-time connection problems caused by configuration errors are most often also persistent. To resolve the issue, verify that the setup has been properly performed. (See section 4)

If the connection worked previously, then the questions you should ask yourself are: How long ago did it work? What has changed? These types of questions will help you to pinpoint what network changes or fix pack level upgrades, application changes or system outages may have taken place between the point in time when it last worked, and now. Identifying what changed and where (i.e. on the client, gateway, or server) helps you to determine where you should focus your energy during investigation.

Question #4: Does it fail for all users / from all clients?

The purpose of this question is to compare with similar working and non-working environments. If the problem occurs only for a subset of the clients connecting to this server, then you should identify the commonalties and differences between these clients, and compare their configurations (DB2 and network). If the problem is occurring across all of your clients, then a simple comparison is not an option for you. If the problem is intermittent but is happening across all of the servers at the same time, then this points to a greater likelihood that the error originates on the server or network, rather than the client. Thus by coupling the information resulting from this question with the answers from the other four questions, you have a better basis for determining where the cause of the problem may lie.

How to confirm your client/server configuration

When to confirm your client/server configuration

To summarize the previous discussion, configuration errors are commonly the source if your communication problem has the following characteristics:

A. It is not an intermittent error;
B. The connection has either never worked, or has changed recently;
C. You can connect successfully locally;
D. You tried creating a loopback connection on the server and it failed. (In this situation you will only be interested in the examples that follow with respect to the "Server" aspect of configuration).

Verification of the setup on the server

The steps to confirm the setup on the server are as follows:

1. Verify the existence of the database by issuing one of the following commands:
   - `db2 list db directory`
   - `db2 list db directory show detail`

   If this is a three-tier environment, you would see the database cataloged as a remote database.

```
            Database X entry:

 Database alias                 = SAMPLE
 Database name                  = SAMPLE
 Database drive                 = C:\DB2
 Database release level         = 9.00
 Comment                        =
 Directory entry type           = Indirect
 Catalog node number            = 0
```

2. Verify that the instance is configured to start listening on the appropriate protocol, via the command `db2set -all`. Look for the parameter DB2COMM.

```
C:\PROGRA~1\SQLLIB\BIN>db2set -all
[e] DB2PATH=C:\Program Files\SQLLIB
[i] DB2INSTPROF=C:\PROGRAM FILES\SQLLIB
[g] DB2SYSTEM=mysystem
[g] DB2PATH=C:\Program Files\SQLLIB
[g] DB2INSTDEF=DB2
[g] DB2COMM=tcpip
[g] DB2ADMINSERVER=DB2DAS00
```

3. Verify that the appropriate protocol-specific parameters are set in the database manager configuration settings, so that DB2 knows what values to listen on. For example, issue `db2 get dbm cfg` and look in the following section:


```
...
NetBIOS Workstation name                (NNAME) =
TCP/IP Service name                     (SVCENAME) = db2cDB2
 APPC Transaction program name          (TPNAME) =
 IPX/SPX File server name               (FILESERVER) =
 IPX/SPX DB2 server object name         (OBJECTNAME) =
 IPX/SPX Socket number                  (IPX_SOCKET) =
...
```

4. If the previous step resulted in a service name instead of a port number for the SVCENAME field in your environment, then confirm that the value listed there is mapped to a unique port number in the operating system's /etc/services file. For example:

```
db2cDB2    50000/tcp   # Connection port for DB2 instance db2inst1
```

Starting DB2 communication listeners on the server

1. Create a backup copy of the db2diag.log, then clear the contents of the original db2diag.log file. Raise the diagnostic level to 4, then stop and restart this DB2 instance, via the following commands:

```
db2 update dbm cfg using diaglevel 4
db2stop
db2start
```

2. Verify that the db2diag.log shows that the listeners for the specific protocol which we are interested have been started successfully.

```
2002-11-24-01.09.46.282346   Instance:lcawley   Node:000
PID:72732(db2tcpcm)   Appid:none
common_communication  sqlcctcpconnmgr_child   Probe:66
DIA3004I Normal shutdown requested for "TCPIP" protocol support.

2002-11-24-01.09.46.784304   Instance:lcawley   Node:000
PID:16744(db2stop2)   Appid:none
base_sys_utilities  stopdbm    Probe:911

2002-11-24-01.09.50.606535   Instance:lcawley   Node:000
PID:23694(db2sysc)   Appid:none
common_communication  sqlcctcpconnmgr   Probe:160
DIA3218I "1" TCP/IP connection manager(s) was/were successfully
started.

2002-11-24-01.09.50.624615   Instance:lcawley   Node:000
PID:23694(db2sysc)   Appid:none
common_communication  sqlcctcp_start_listen   Probe:80
DIA3000I "TCPIP" protocol support was successfully started.

2002-11-24-01.09.51.652177   Instance:lcawley   Node:000
PID:16748(db2star2)   Appid:none
base_sys_utilities  startdbm   Probe:911
```

3. If you had not performed one of the necessary setup steps, you would receive the following error when you performed the `db2start` command:

```
SQL5043N  Support for one or more communications protocols
failed to start successfully. However, core database manager
functionality started successfully.
```

4. This would be accompanied by specific error messages in the db2diag.log file if DIAGLEVEL was set to 4 at the time. For example, if I had not set the SVCENAME parameter in the database manager configuration settings, I would receive the following error in the db2diag.log:

```
2002-11-24-01.13.13.068436   Instance:lcawley   Node:000
PID:20222(db2sysc)   Appid:none
common_communication  sqlcctcpconnmgr   Probe:50
DIA3200E The SVCENAME parameter in the database manager
configuration file is not configured. Update the SVCENAME
parameter using the service name defined in the TCP/IP
services file.
```

5. Once you have confirmed that the DB2 protocol-specific listeners are started successfully for the instance, don't forget to downgrade the diagnostic level, so that you do not suffer performance problems. Returning it to the default diagnostic level of 3 is done as follows:

```
db2 update dbm cfg using diaglevel 3
db2stop
db2start
```

Verification of the setup on the client

1. Confirm the values that you used in your cataloging of the remote server's instance by executing `db2 list node directory` or `db2 list node directory show detail`:

```
C:\>db2 list node directory show detail

 Node Directory

 Number of entries in the directory = 1

Node 1 entry:

 Node name                      = MYNODE
 Comment                        =
 Protocol                       = TCPIP
 Hostname                       = myserver
 Service name                   = 50000
 Remote instance name           =
 System                         =
 Operating system type          = None
```

The "Service name" value in this output should match the port number defined in the server instance's database manager configuration SVCENAME setting. If a service name is used instead of a port number, it is important that the port number to which the service name on the client maps to should match the port number used on the server.

2. Verify that you can ping the Hostname value exactly as it appears in the results from step (1).

3. Another test that you can perform is to use the `telnet` command to check whether there is something (in this case, we hope a DB2 instance) listening on a particular port number on a particular server. For example: `telnet myserver.ibm.com 50000`. Telnet does not have to be enabled on the server. If there is indeed something listening on that port then we would expect to see that the telnet window will open, but then hang. This means that we have indeed reached that server, and that something is indeed listening on that port. If, on the other hand, we receive an immediate error, then we know that either:
    a. We have the wrong hostname or IP address values. If you used a hostname here, try using the IP address, in case it is a hostname resolution problem;
    b. We have the wrong service name or port number. If you used a service name, try using the port number instead, in case the service name was not mapped correctly in the /etc/services file on this client;
    c. There is a firewall between the client and server, and it is not allowing communication on this port. Verify with your network administrator whether or not this is the case;
    d. DB2 on the server is not listening on this port. Refer back to the beginning of this section for information on how to confirm the setup of the server.

4. If the telnet and ping tests worked, confirm that the proper database values appear in the database and DCS database catalogs, as appropriate. Errors in these catalogs do not usually result in communication errors, however.

At this point you have confirmed that the configuration of both the client and the server are correct from the DB2 point of view. If you continue to receive a communication error, these are situations where we would then pursue obtaining diagnostics like db2 traces.

Diagnostics to collect

Where to look for assistance (connectivity-specific)

A subset of the protocol-specific error codes are listed in the appendix of the DB2 *Message Reference* manual. Further information may also be found as follows:

TCP/IP:/usr/include/sys/errno.h on UNIX®; nerrno.h on OS/2; winsock.h on Microsoft Windows. The winsock.h file may not be installed on your system if you do not have a Microsoft Windows development environment installed.

APPC: Refer to the documentation for your APPC product.

NETBIOS: Refer to the *LAN Technical Reference: IEEE 802.2 & NetBIOS APIs*

IPX/SPX: Refer to the error values in tiuser.h on OS/2 or UNIX (in /usr/include/sys); winsock.h on Microsoft Windows.

Data to collect

The data most often requested for persistent communication errors are:

- dbm cfg from the server
- Catalogs from both the client and the server
- Operating system and DB2 version and FixPak level info from both

This information can be obtained by issuing the following command on the client and on the server:

```
db2support <output path>
```

The information returned by this tool will be most complete when run by a userid with SYSADM authority.

If the DB2 Support Team has requested a db2 trace for a communication error, we will tell you how we would like the trace to be taken. In general, we want the trace taken in the simplest possible recreation scenario. For example, if the problem can be recreated solely on the server (either via local connection or loopback connection), then we would want the trace to be taken on the server. If, on the other hand, the problem recreation requires both a client and a server, then we would want traces to be taken simultaneously on the client and the gateway or server. This allows us to follow the flow of informatoin from the client to the server, and helps us to determine where the breakage or error is originating.

Summary and feedback

What you should now know

By this point, you should understand the importance of knowing your environment, and how to use this knowledge to isolate the source of communication errors. You should be familiar with the use of the PCT tool, as a method of verifying communication links prior to involving DB2 in the picture. You should know how to use four simple questions to help you determine the nature and effects of the communication error you are receiving, and how to in turn use that information to further narrow down the cause of the error. You should have firsthand knowledge of how to verify the configuration of a client and server connection via TCP/IP, as well as how to create a loopback connection on a server.

For more information

Refer to the *DB2 Troubleshooting Guide*, and *Installation and Configuration Supplement*.

Feedback

Please take a moment to give us your thoughts on this tutorial by completing the feedback form on the DB2 Technical Support site at http://www.ibm.com/software/data/db2/udb/winos2unix/support .