

SQL*Net™ PERFORMANCE TUNING UTILIZING UNDERLYING NETWORK PROTOCOL

Gamini Bulumulle
ORACLE CORPORATION
5955 T.G. Lee Blvd., Suite 100
Orlando, FL 32822 USA

Summary

Oracle client/server architecture is a method of separating a data processing system into two parts. One is the *client*, executing application software which issues data requests to the *server*, executing the database application software which responds to client requests and controls the database as required.

Oracle client/server systems will employ SQL*Net™, an interface between Oracle application software and the underlying network protocol (UNP). SQL*Net allows Oracle products to access, modify, share, and store data on heterogeneous computing platforms in a variety of networking environments.

This paper discusses performance optimization and tuning of SQL*Net based on an arbitrary UNP which could be TCP/IP, IPX/SPX, or DECnet, among others. SQL*Net performance may be maximized by synchronization with tunable parameters of the UNP, for example, buffer size.

Total SQL*Net transaction performance can be broken down into components of connect time and query time. Connect time can be minimized by calibration of tunable parameters of SQL*Net and the UNP when designing and implementing networks. Query time is typically affected by database tuning parameters which are outside the scope of this paper. However, database tuning parameters, which impact network performance, are discussed here.

Test results comparing Oracle client/server performance on TCP/IP, IPX/SPX, and DECnet are presented, as well as standards for optimizing transaction performance with respect to connect and query time.

The Oracle stack

This paper discusses the performance optimizing and tuning of SQL*Net as well as the interdependent UNP. Performance of a client/server application can be optimized by expediting connect and query times between client and server and reducing network traffic.

When configuring client/server applications, performance is impacted by

- Configuration parameters of the application, i.e., SQL*Plus or Oracle Server:
- Parameters of SQL*Net
- Parameters of the UNP

In the following illustration, typical upper layer protocols (ULPs) could be TCP/IP, IPX/SPX, or DECnet; lower layer protocols (LLPs) could be Ethernet, Token Ring, or FDDI.

The Oracle client/server model can be mapped into the Open System Interconnection (OSI) reference model. Oracle client applications such as SQL*Plus or SQL*Forms, and server applications such as the Oracle relational database management system (RDBMS) reside at layer seven of the OSI model; SQL*Net at layers five and six; the ULP at layers three and four; the LLP at layer two; and the physical layer at one. In this discussion, application software resides at the top of the stack. Overall application performance is based upon the performance of the lower three layers as well as variable external factors such as network traffic.

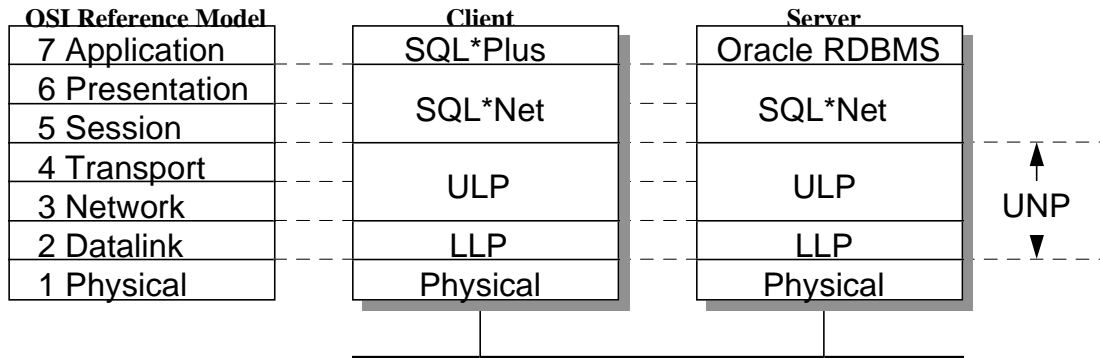


Figure 1 Oracle client-server model

The “stack” paradigm may be applied to SQL*Net performance, which depends to a great extent on the performance of the lower layers. Therefore, when designing or implementing Oracle client/server systems, it is vital to take into consideration tunable parameters of the underlying layers in order to optimize the performance of SQL*Net.

SQL*Net performance and tuning

For this discussion, SQL*Net performance and tuning analysis is based on these two categories:

- SQL*Net performance
- SQL*Net tuning

SQL*Net performance

Performance of SQL*Net is based on several factors. These will be discussed in this section. Consider the data communication transaction resulting from a simple SQL*Plus statement:

The SQL*Plus client application initiates a network message as a result of the above statement. The message is received by the server, data is retrieved, and returned through the network to the client:

```
SQL> select * from dual;
D
-
X
SQL>
```

Performance may be rated by the difference between the time the client application presents a communication request to the client SQL*Net (t_1) to the time the client SQL*Net returns the response to the client application (t_2). Referring to **Figure 1**, ($t_2 - t_1$) is the time required for data to be propagated through client layers 6 through 1, be transported across the network medium, be propagated through server layers 1 through 6, plus the symmetric return trip.

($t_2 - t_1$) may be further broken down into connect time and query time. *Connect time* is the round-trip time spent communicating data between client and server application layers, and *query time* is the time spent processing the data by the server.

Thus,
$$\Delta t = t_2 - t_1 = \text{connect time} + \text{query time} \tag{1}$$

Factors affecting connect time

Connect time is based on various external factors as well as the statuses of certain Oracle run-time options and helper utilities.

External Factors	Oracle options and utilities
<ul style="list-style-type: none"> • Use of domain name service • Network topology • Network throughput (i.e., data rate) • Number of “hops” (bridges, routers) between client and server • Network contention, if applicable • Response time • Heterogeneous network protocols 	<ul style="list-style-type: none"> • Prespawn processes • Multi-threaded server (MTS) vs dedicated connections • Size of Tnsnames.ora file • Status of SQL*Net tracing • Status of security features

Prespawn processes—Prespawn dedicated server processes provide a faster connection to the database by eliminating the time required to spawn a process for each connection request.

MTS vs dedicated connections—MTS has its own dispatcher already existing. Dedicated environment needs to create a processes. This make it a little slower.

Size of the **Tnsnames.ora** file— The **Tnsnames.ora** file, which resides on the client, is significant for applications using SQL*Net. The size of this file may be directly related to connect time. When a client application initiates a transaction to retrieve data from a server, the entire **Tnsnames.ora** file is read.

Example:

```
$ sqlplus uid/passwd@alias_name
```

alias_name is stored in the Tnsames.ora file. Thus, a portion of connect time is determined by the size of Tnsnames.ora. Instead of reading the entire file and scanning for the relevant entry, it is better to implement an indexing method.

SQL*Net tracing—If SQL*Net tracing is turned on, then every client/server connection generates a trace file. These files are usually large. The size of the file depends on the level of tracing. Since tracing generates a trace file, it slows down the connect time.

Security features—Implementation of security features such as encryption/decryption algorithms increase processing time at both ends of each secure transaction.

Factors affecting query time

Once the connection is made, query time is the amount of time required to retrieve data from the database. Query time is impacted by the following factors:

- Indexing
- Array size

Indexing

Such factors affect performance at the database level. Since this paper focuses on network performance, discussion is limited to array size.

Array size

The size of the array_size parameter impacts performance. For example, in SQL*Plus, array_size is defined by the SET command, e.g.,

```
SQL> set array_size value
```

value determines the number of rows (called a batch) that SQL*Plus will fetch from the database at one time. value may range from 1 to 5000. A large value increases the efficiency of queries that fetch many rows, but requires more host memory.

By calibrating the `array_size`, it is possible to distribute the time required to query the records rather than fetching them all at once, thus decreasing the *perceived* query time. Note that the total time to query the records in smaller groups may be greater than the total time to query the records all at once. Computational overhead to access the database will be repeated for each call to the database when `array_size` is less than the number of records required to be fetched. If the `array_size` parameter is large, then the impact of the overhead is minimal, but additional time is required to retrieve the batch. If `array_size` is smaller, then the frequency that the overhead impacts the database is greater, but data retrieval time per batch is smaller.

Put another way, when retrieving an arbitrary number of rows, smaller `array_size` reduces fetch time but increases overhead, whereas larger `array_size` increases fetch time but reduces overhead. Overall, a larger `array_size` produces better results.

Referring to expression (1), there are tradeoffs between connect time and query time. Using a larger `array_size` might optimize query time, at the expense of connect time and overall performance. It is important to determine the optimum batch size, which is a product of `array_size` and row length. Row length in turn is a function of the type and amount of data (e.g., VARCHAR2, LONG) in a table.

Session Data Unit (SDU) parameter

If `array_size` is set to a higher figure based on row data type, the application passes a large amount of data to SQL*Net. The amount of data able to be processed by SQL*Net (refer to **Figure 1**) depends on the SQL*Net buffer size. The SQL*Net buffer is defined by the Session Data Unit (SDU) parameter. For SQL*Net version 2.3.x and above, the default size of the SDU parameter is 2 KB (configurable up to 32 KB); for versions 2.3 and below, the default SDU is also 2 KB (the maximum configurable size). As a SQL*Net connection is established, the client and server negotiate the size of the SDU that will be used. If the SDUs of the client-side and server-side differ, the smaller of the two will be selected. This “decision” is made by the server-side SQL*Net.

If the SDU parameter is smaller than the application fetch size, fragmentation could occur. If SDU is larger than the application fetch size, there is no fragmentation, and the entire packet can be sent across the network (assuming ULP and LLP buffer sizes are large enough to handle it).

Again, the `array_size` is the number of rows that Oracle will fetch before it passes them to the server SQL*Net to be returned to the client. This will affect SQL*Net packet sizes throughout the communication stream.

Figure 2 Syntax SDU in `Tnsnames.ora` file:

```
EOUG=
  (DESCRIPTION=
    (SDU=2048) service layer buffer size
    (TDU=1024) transport layer size
    (ADDRESS=
      (PROTOCOL=TCP)
      (HOST=ORLSUN9)
      (PORT=4446)
    )
    (CONNECT_DATA=
      (SID=V7321)
    )
  )
```

Figure 3 Syntax SDU in `Listener.ora` file

```
LISTENER=
  (ADDRESS_LIST=
    (ADDRESS=
      (PROTOCOL=TCP)
      (HOST=ORLSUN9)
      (PORT=4446)
    )
  )
STARTUP_WAIT_TIME_LISTENER=0
CONNECT_TIMEOUT_LISTENER=10
TRACE_LEVEL_LISTENER=OFF
SID_LIST_LISTENER=
  (SID_LIST=
    (SID_DESC=
      (SDU=8192)
      (SID_NAME=V7321)
      (ORACLE_HOME=ORACLE/7321)
    )
  )
)
```

Example:

Assume the SDU is 2K. If the `array_size` was set to 3 and the first 6 rows of data are the following sizes (in bytes): 1511, 410, 730, 300, 200, 500

The Oracle server would first request the server side SQL*Net to send 2651 bytes (the first three rows), then 1000 bytes (the last three rows). Oracle server would send the following datagrams:

Datagram	Size	Data bytes	SQLNet Header - bytes
1	2048 (SDU)	2038	10
2	623	613 remaining	10
3	1010	1000 requested	10

Relationship between SDU and Maximum Transfer Unit (MTU) parameters

The MTU defines the buffer size of UNP, specifically with TCP/IP. The relationship between SDU and MTU parameters can be summarized as follows:

If SDU = MTU this is the ideal situation; no fragmentations occur
 else if SDU > MTU there is fragmentation
 else SDU < MTU where performance does not increase

Note: the above three conditions are met considering there is enough space left for UNP header information.

Example:

Assume the ULP is TCP/IP and the MTU parameter (buffer size) is set to 1500. Packet #1 is 2048 bytes (condition: SDU > MTU), which cannot be “absorbed” by the ULP because of ULP buffer size limitations. As a result, fragmentation occurs and performance suffers.

Example:

TCP/IP-level fragmentation:

2048 SQL*NET buffer size

1500 TCP/IP buffer size

This combination will generate two SQL*NET packets. Packet #1a is 1500 (1460+40) bytes and packet #1b is 628 (588 + 40) bytes. As a result of this fragmentation, the amount of traffic passed to the LLP is increased by a multiple of two. When these packets go through the datalink layer, more data is prepended (e.g., Ethernet, 14 bytes). Theoretically, at the bottom of the client stack, the size of the two packets are:

$$1500 + 14 = 1514 \text{ packet1a}$$

$$628 + 14 = 642 \text{ packet1b}$$

Now consider packet#2 (SDU < MTU). Since the size of this packet is 623 bytes, less than the MTU size (1500 bytes), there is no fragmentation. However, increasing SQL*NET packet size it is possible to increase performance as a larger packet transform across the network.

packet#2 ---> 623 (data) + 40 (TCP/IP header) + 1 padding (byte) + 14 (Ethernet Header) = 678 data (bytes)

Now consider the ideal situation where SDU = MTU. In this situation, there is no fragmentation as the buffer sizes are synchronized. This is the optimum situation.

Sql*Net Tuning

As discussed, the performance optimization means reducing network traffic, which may be achieved through the *tuning* process.

According to **Figure 4**, the Oracle server application passes a *batch* of data to SQL*Net, where a 10-byte control header (H_S) is prepended, forming a *frame* which is passed to the ULP. The ULP prepends its header H_{ULP} , the size of which depends on the protocol used TCP/IP¹, for example, uses a 40-byte header²; IPX/SPX, a 30-byte header³, forming a *datagram* which is passed to the LLP. The LLP prepends its header H_{LLP} , the size of which again depends on the protocol used Ethernet, for example, uses a 14-byte header⁴, forming a *packet* which is passed to the physical layer for transmission.

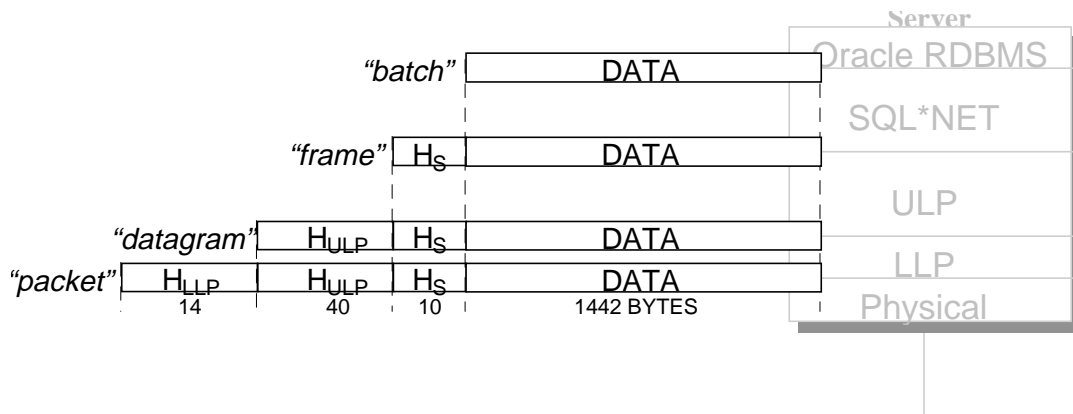


Figure 4 Data flow through the server network stack.

Ideally, if the data buffers of SQL*Net, the ULP, and the LLP are synchronized, then fragmentation is minimized or eliminated as data flows from the application layer to the LLP.

Example. Assume the SDU is 2K, the ULP (TCP/IP) MTU is 1500 bytes and LLP (Ethernet) buffer is 1506 bytes. The application passes 1442 bytes of data to SQL*Net, which prepends a 10-byte header, producing a frame of 1452 bytes. SQL*Net in turn passes the frame to the ULP, which prepends a 40-byte header, producing a datagram of 1492 bytes. ULP then passes the datagram to the LLP, which prepends a 12-byte header, producing a packet of 1506 bytes. The batch has successfully passed through the client stack without fragmentation.

In this example, note that since each succeeding lower layer buffer is large enough to absorb the data received from its respective upper layer, there is no fragmentation. This is the ideal situation. In practice, this is seldom possible due to incompatibilities between buffer sizes of the layers. When data flows between layers of incompatible buffer sizes, fragmentation occurs, and as a result, extra network traffic is generated. With this in mind, components of the stack may be tuned in order to minimize fragmentation, which reduces network traffic and thereby increases performance.

Tests

Test conditions:

- LAN and WAN
- Default parameters, then by varying values of parameters such as SDU, TDU and array_size.
- Set TDU = 1024 and change SDU = 1K, 2K, 4K and 8K.
- Set TDU = 2048 and change SDU = 1K, 2K, 4K and 8K.
- Change the array_size to 1, 2, 5, 10, 25, 50, 100, 150, 1000
- Client application: SQL*Plus 3.3.2.0.2
- Client-side SQL*Net 2.3.2.1.4; server-side SQL*Net 2.3.2.1.0.
- Server O/S - Sun Solaris 2.5
- Client O/S - Windows NT 4.0
- RDBMS version - 7.3.2.1.0
- Created three tables:
 - stats1 (4096 rows, 20 B/row, VARCHAR2)
 - stats2 (4096 rows, 100 B/row, VARCHAR2)
 - eoug (410 rows, LONG).
- Screen output disabled with `set termout off;`
- Timing enabled with `set timing on;`
- Turn on SQL tracing using:


```
alter session set sql_trace=true;
```
- Network protocols TCP/IP, SPX/IPX and DEC-Net.
- Testing on LAN - client/server on the same subnet.
- Testing on WAN - client/server on different subnet.
- Each test was repeated three times and the results averaged.
- Ethernet was used as the LLP for all tests.

Test Methodology

Since the size of the SDU parameter is negotiated between the server and the client, the size of the parameter on the server side was set to 8 KB and did not change as the client SDU was varied from 1 KB, 2 KB, 4 KB to 8 KB. When the TDU and `array_size` parameter were set to default values, the following results were observed with respect to 'connect time' and 'packet size'.

Initiate the following command from the client and captured the transmitting time ($t_{c \rightarrow s}$) from client to server using the LAN analyzer.

```
$ sqlplus system/manager@test1;
SQL>
```

Then fetch/retrieve data from the server table using the following command.

```
SQL> select * from stats1;
```

Finally, the data transmission from server to client and the transmitting time ($t_{s \rightarrow c}$) was captured using a LAN analyzer. The total connect time is as follows:

$$\text{total_connect_time} = t_{c \rightarrow s} + t_{s \rightarrow c}$$

Next on the client setting `array_size` and TDU parameters as constants, change SDU parameter from 1K, 2K, 4K and 8K and the following results were obtained.

Results/Observations

Changing the SDU parameter did significantly improve connect time ($t_{s \rightarrow c}$). This can be attributed to the amount of data being retrieved remaining constant, but the size of the packets being varied.

Table 1: Table Stats2 row size =100

Batch size = 1500 bytes, TDU=default size Array size=15 (default) and client SDU

SDU (KB)	TCP/IP Connect Time	IPX/SPX Connect Time	DECnet Connect Time
1	211.9547	152.177	164.813
2	133.3079	152.093	164.501
4	128.0889	151.997	164.721
8	124.5137	152.321	165.008

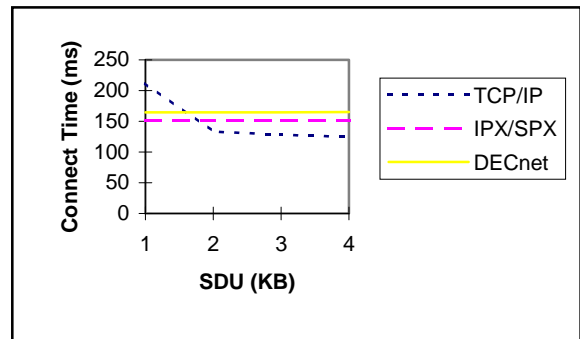


Table 2: Table Stats2 row size =100

Batch size = 1500 bytes, TDU=1024 size Array size=15 (default) and client SDU

SDU (KB)	TCP/IP Connect Time	IPX/SPX Connect Time	DECnet Connect Time
1	162.6090	153.003	163.792
2	159.7580	151.897	164.036
4	161.8337	152.349	164.492
8	160.5553	152.102	164.123

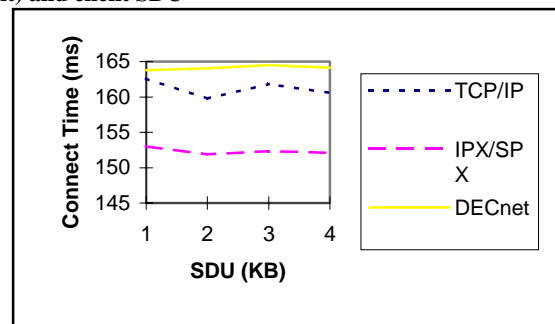
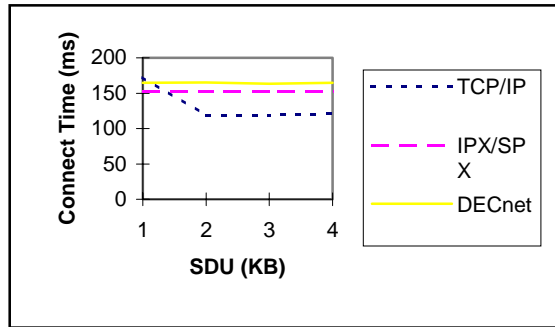


Table 3: Table Stats2 row size =100

Batch size = 1500 bytes, TDU=2048 size Array size=15 (default) and client SDU

SDU (KB)	TCP/IP	IPX/SPX	DECnet
	Connect Time	Connect Time	Connect Time
1	172.2118	152.001	164.514
2	119.0573	151.923	165.121
4	118.9678	152.147	163.447
8	120.9134	152.391	164.823



Based on these three performance curves, the best performance is at a TDU size of 2048 B. Also, further increments of SDU after 2 KB have negligible effect on the connect time.

The above three graphs provide performance comparison among the three network protocols in a client/server environment and by calibrating all the variables discussed in this paper, it was observed that the protocol which contains the largest buffer size provides the best performance. Protocols SPX/IPX and DECNet maximum buffer sizes are 576 bytes and whereas TCP/IP 1500 bytes. SDU parameter is always greater than SPX/IPX, DECNet buffer sizes, so by changing SDU parameter did not gain in connect time.

It may also be observed from the TDU=1024 B curve that when SDU ≥ TDU, no performance gains are realized as the TDU becomes a bottleneck, causing numerous smaller packets to be generated. Since these protocols, being a connection-oriented protocol, generates a corresponding number of ACK packets causing the network traffic increases proportionally.

Table 4: Table Stats1 row size =20

Batch size = 300 bytes, TDU=default size Array size=15 (default) and client SDU

SDU (KB)	TCP/IP	IPX/SPX	DECnet
	Connect Time	Connect Time	Connect Time
1	70.2852	70.281	71.237
2	70.1914	69.953	72.112
4	70.3982	69.963	69.923
8	70.2184	69.852	70.167

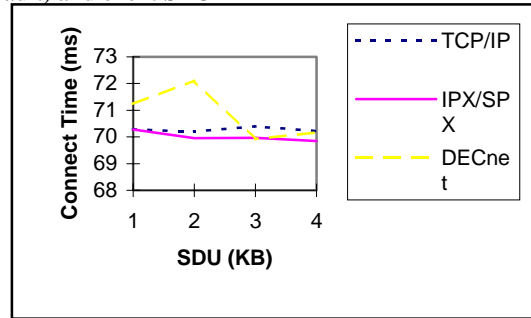


Table 5: Table Stats1 row size =20

Batch size = 300 bytes, TDU=1024 size Array size=15 (default) and client SDU

SDU (KB)	TCP/IP	IPX/SPX	DECnet
	Connect Time	Connect Time	Connect Time
1	70.2717	70.212	71.167
2	71.6192	71.375	73.578
4	69.7773	69.351	69.306
8	70.2174	69.851	70.166

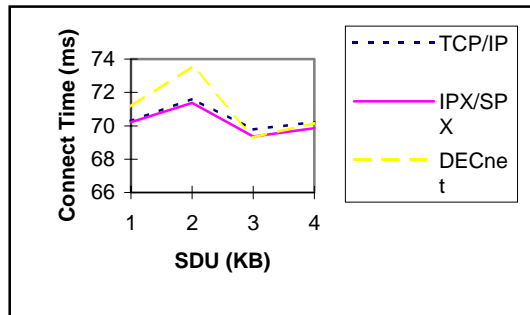
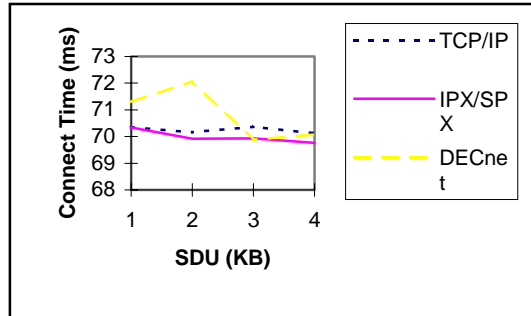


Table 6: Table Stats1 row size =20

Batch size = 300 bytes, TDU=2048 size Array size=15 (default) and client SDU

SDU (KB)	TCP/IP	IPX/SPX	DECnet
	Connect Time	Connect Time	Connect Time
1	70.3445	70.340	71.297
2	70.1581	69.919	72.077
4	70.3625	69.933	69.887
8	70.1279	69.761	70.076



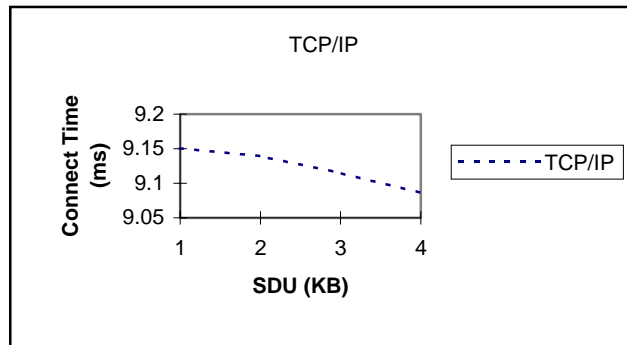
According to above results the batch size was 300 bytes ($300 \ll \text{SDU}$) and not enough data fill SQL*NET and/or UNP buffers. Therefore, the connect time was consistent for each protocol and by comparison performance of each protocol closely match.

Relationship between Long Data Type and SDU.

Datatype LONG

Table 7 Packet size=131 B

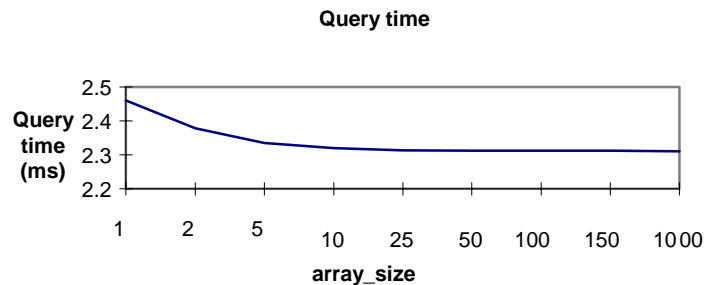
SDU (KB)	TCP/IP
	Connect Time
1	9.1504
2	9.1395
4	9.1149
8	9.0861



According to the above test results, if datatype is LONG then synchronizing buffer sizes has no effect on performance. It can also be observed that changing SDU, TDU, or array size has no effect on packet sizes and the packet size was approximately 131 bytes. This may be the reason why tuning parameters have no effect. Further studies are necessary to investigate this. Repeated the same test for IPX/SPX and DECnet protocols had similar results.

Table 7 Query Time

array_size	Time (ms)
1	2.4602
2	2.3780
5	2.3346
10	2.3202
25	2.3126
50	2.3122
100	2.3120
150	2.3117
1000	2.3105



When retrieving an arbitrary number of rows, smaller array_size reduces fetch time but increases overhead. Larger array_size increases fetch time but reduces overhead. Overall, a larger array_size produces better results.

Recommendations/Standards

1. synchronizing buffer sizes
2. application must have the capability to determine the Array Size
3. implementing indexing method to read Tnsnames . ora file

Conclusions

Based on the above results it could be observed that by changing the SDU parameter did make a noticeable significant difference in transmitting time and also not based on ULP. A LAN analyzer reports the size of the data packets to be 350 bytes. The default `array_size` is 15 and each row in the 'stats' table was 20 bytes.

Example:

$[15 (\text{array_size}) * 20 (\text{bytes row size})] + 10 (\text{bytes SQL*Net header}) + 40 (\text{bytes TCP/IP header}) = 350$
bytes- (2)

Since the minimum value of SDU (1 KB) is greater than 350 bytes, changing the SDU value did not affect the lower levels, which were able to absorb all the data without fragmentation. A product of row size and `array_size` plus SQL*Net header greater than 1 KB would cause fragmentation. Calibrating SDU parameter does not improve performance (connect time).

Example:

If row size is 100 bytes then according to (2) a frame of 1550 bytes would be produced. If the SDU is 1K, then the results would be $((950 \text{ data} + 10 \text{ SQL*Net header} + 40 \text{ TCP/IP header} = 1000) + (550 \text{ data} + 10 \text{ SQL*Net header} + 40 \text{ TCP/IP header} = 600)) = 1600$ bytes. It is here that the fragmentation occurs.

Based on the above results, it could be concluded that implementing a network application (client/server) is a complex tedious process since the efficiency of the application depends on many other external factors.

It is important for a network application developer to understand the 'stack' paradigm in order to design an efficient client/server application. Also, it is very important in order to have a efficient environment, select proper ULP, LLP, and network hardware implementation.

It could also be concluded that performance is based on raw length of a table as well as datatype. It was observed that connect time and query time are independent of each other but total performance is based on these two components.

Synchronizing SQL*NET, ULP and LLP buffer sizes produces the optimum client/server performance. SQL*NET should be intelligent enough to adjust its buffer size automatically according to UNP buffer size. UNP should have the capability to synchronize its buffer with LLP. Having two different parameters (SDU/TDU) complicates the situation. Since whatever the smaller of the two determines the buffer size, either SDU or TDU becomes the bottleneck.

These tests results do not reflect the testing on Wide Area Networks (WAN) and are left for future discussions. Future studies are necessary to investigate the effectiveness of LONG data type on client/server applications. Also observed is that row size does not have one to one relationship with the connect time. Please note that table STATS1 contains 100bytes@row and STATS2 20bytes@row. The row length proportion is 5:1 but connect time based on these results (table numbers) approximately 2:1.

References

1. Postel, J.B. Transmission Control Protocol. 1981 September; 85 p. (RFC 1123)
2. Branden, R.T. ,ed. Requirements for Internet hosts-application and support. 1989 October; 98 p. (RFC 793)
3. Naugle, M.; Network Protocol Handbook. 1994, 186 p.
4. Nassar, D.J., Ethernet and Token Ring Optimization. 1996, 513 p.5.