VLATKO LIPOVAC, Ph.D.
E-mail: vlatko.lipovac@unidu.hr
VEDRAN BATOŠ, Ph.D.
E-mail: vedran.batos@unidu.hr
University of Dubrovnik
Dpt. of Electrical Engineering and Computing
Branitelja Dubrovnika 29, HR-20000 Dubrovnik,
Republic of Croatia
ANTUN SERTIĆ, Ph.D.
E-mail: antun.sertic@fpz.hr
University of Zagreb,
Faculty of Transport and Traffic Sciences
Vukelićeva 4, HR-10000 Zagreb, Republic of Croatia

Information and Communication Technology
Preliminary Communication
Accepted: May 14, 2008
Approved: Dec. 22, 2008

TESTING APPLICATION (END-TO-END) PERFORMANCE OF NETWORKS WITH EFT TRAFFIC

ABSTRACT

This paper studies how end-to-end application performance (of Electronic Financial Transaction traffic, in particular) depends on the actual protocol stacks, operating systems and network transmission rates. With this respect, the respective simulation tests of performance of TCP and UDP protocols running on various operating systems, ranging from Windows, Sun Solaris, to Linux have been implemented, and the differences in performance addressed focusing on throughput and response time.

KEY WORDS

end-to-end application performance, quality-of-service, EFT traffic

1. INTRODUCTION

1.1 Managing end-to-end application-level performance of multiservice networks

Network services increase in number, sophistication and real time dependency, Figure1.

Therefore, the enterprise-computing environment has become a fantastically complex entity. Because of the instability of growing networks, network managers are mostly in "survival mode" as their main concern is configuring the network and keeping it from going down by taking care of all kinds of network, system, application, and security components. Therefore, networking staff mostly think in terms of individual routers, switches, and LAN segments. The challenge is to configure, benchmark, and integrate all these components without getting distracted from their original mission of supporting and enhancing core business processes. However, network managers should also be thinking of how these pieces fit together to deliver increased productivity. Simply put, there are too many variables for IT departments to rely exclusively on element-centric tools, so that the number of devices, applications, protocols, operating systems, and technologies has forced IT managers to spend so much time looking at the trees that they cannot possibly see the forest. In addition, applications that run in today's complex network environments often encompass numerous elements across the network, whose cumulative impact can be quite severe. Without the ability to measure end-to-end service performance, this kind of degradation never shows up on an alarm console. In this sense, it is necessary to have the forest-level monitoring application that will enable IT departments to view the network as their customers see it. So, in contrast to today's still dominant element-centric approach to managing a network, observing end-to-end performance of the network, and how that affects the end user, is actually what IT staff should be concerned

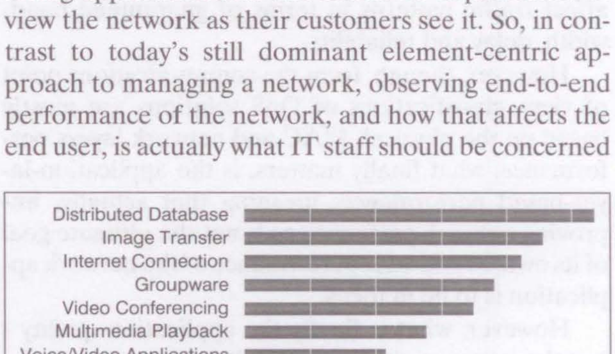

Figure 1 - Variety of network services

with, as business managers care about how quickly a database query is answered, or what is the responsiveness of the applications they use daily. Moreover, network performance that impacts application performance goes further, to impact overall business performance, so that network problems can have severe negative impact on overall business performance and results.

1.2 Quality of service and application performance

Quality-of-service (QoS) is a very popular and overloaded term having as many interpretations as interpreters. It is very often looked at from different perspectives by the networking and application-development communities. In networking, the term "QoS" refers to the ability to provide different treatment to different classes of traffic. The primary goal is to increase the overall utility of the network by granting priority to higher-value or more performance-sensitive flows. With this respect, QoS is a term which qualifies the performance of the entire network, and it is thus of vital importance that all elements of the network function at the satisfactory level. Specifically, ITU-T recommendations define QoS as "the collective effect of service performance, which determines the degree of satisfaction of a user of the service".

Introduction of multiservice networks demands more intelligent control over network usage and more efficient application development practices that enable achieving QoS goals. As applications over the network increase, so does the need to diagnose performance at the application level, and network designers need to implement a proper QoS technique, knowing the differences between the techniques and how they affect traffic patterns in terms of guaranteed bandwidth, delay and reliability.

However, though, from the communications point of view, classifications of QoS solutions are mostly based on the physical, MAC and network layers performance, what finally matters, is the application-layer-based performance, meaning that actually, improving network performance is not the ultimate goal of its own, as end-user performance of the network application is to be in focus.

However, what is finally the application quality - just the response-time to "enter"?

The term "application quality" is too vague to be deterministically defined. The reason is that the factors that influence quality are very "fuzzy". Network QoS allows for the definition of quality metrics based on variety of parameters, but such QoS models are engineered using network-centric quality parameters (available bandwidth, delay, jitter etc.). Application developers and users, on the other hand, require quality models that are geared to their needs, and that are expressed by different performance characteristics such as response time, availability, throughput, predictability, and consistent perceptual quality. These are metrics that define what is called application quality. Such factors include user's expectations and experience, the task of the application, and whether the application delivers the expected levels of performance. Furthermore, other factors, like charging for the use of the network resources or the service, also influence the application quality.

In this paper, we focus on the transaction-intensive network application traffic profiles, such as the ones associated with Electronic Financial Transactions (EFT), in particular, where proper network performance is a mission critical to the state-of-the-art Integrated Transaction Management (ITM) solutions.

2. TESTING APPLICATION PERFORMANCE

2.1 Test methodologies

Regardless as to whether the focus is on studying topologies, traffic characteristics or interactions with other protocols, the methods of studying telecommunication networks can be categorized into three main types: mathematical modelling, real-life measurements, and simulation (and/or emulation).

Mathematical modelling of the problem provides exact results, but the number and the complexity of application performance calculations grow drastically as the network complexity increases. It provides deeper understanding of fundamental rules of the studied phenomenon for networks with a relatively small number of input data, but the downside of this method is that it can lead to oversimplifying the model.

Simulation and emulation of networks is a widely used testing method today, but the methodology and testing approaches have not yet been fully defined, even though some literature offers certain frameworks that allow us to come to usable results. The main disadvantage of the simulation and emulation method is that both may ignore or omit real life occurrences. However, their main advantage comes from processing large amounts of data and testing the network reactions to different input parameters. For example, protocol simulation often ignores details about the implementation of the protocol on the operational system or the information content of the packets, while it focuses mostly on the algorithms and traffic quality parameters. Simulation has proven to be a powerful tool for checking the results of mathematical modelling and its significant advantage over real life measurements lies in the possibility to test the state of the network at any network point. The disadvantage of this method is that it does not take into consideration all the effects in the test environment, which results in a narrow application field of certain tools and variable reliability of the results. Consequently, the results of the simulation method have to be used as good reference for conducting final conclusions, in combination with modelling and measurement results.

Finally, measuring real parameters of network QoS enables detailed analysis, as well as further understanding of what is complying to regulations or not. Using specialized hardware or software, engineers gather relevant data about the state of some part of the network.

With this respect, at last, we point out that when dealing with network performance, one should exploit the *synergy* among analytical models, program simulations, and experimental testing, as a methodical framework on top of which practical engineering approach can be built.

2.2 Experimental system

The software application used was Agilent Application Analyzer to explore the basic application performance issues by driving the TCP and UDP protocol stacks and measuring the round-trip response time of three typical network transactions. We identified the difference among the performances of these protocol stacks throughout various operating systems and Ethernet network transmission rates of 10 and 100 Mbps. (While higher Ethernet speeds (Gbps) are still not widely available on desktop, another reason for not considering them in this paper was that differences between protocol performances might be more visible if not hidden by waste available bandwidth, though, on the other hand, the choice of operating system becomes more pronounced with faster networks, whose bandwidth cannot be exhausted by a single connection, as it is the case in 10 Mbps networks.)

Application Analyzer generates network traffic between pairs of agents and observes the performance of the traffic whose patterns were tailored to match the traffic of the real applications of interest – the typical EFT ones.

The remote agent programs were created and operated from the console, named Simulation Center, Figure 2.

Creating a test consisted in deciding which of the available software-based distributed active agents (and on which computers) to use as an endpoint pair, Figure 3. A set of pre-built application scripts provided standard performance benchmarks and emulated common end-user applications whose activation comprised specifying the network addresses and the network protocol to use between them, as well as the type of application (data traffic tests that include: loss, delay, throughput, jitter, out-of-order, QoS (UDP, TCP, RTP, VoIP), ICMP etc.) to emulate between the endpoints, in order to measure the performance of network, applications and services from "anywhere to anywhere" (over almost any transport mechanism: LAN, ATM, Frame Relay, wireless), for each user and in real-time. Though Application Analyzer supports tests with multiple concurrent connections between various endpoints, we limited our testing to just one connection between endpoints at a time.

Application Analyzer agents were installed on several operating systems[1], supporting various network protocols, as well as TCP and UDP transport protocols. On Windows platforms, Application Analyzer issues calls to the WinSock application programming

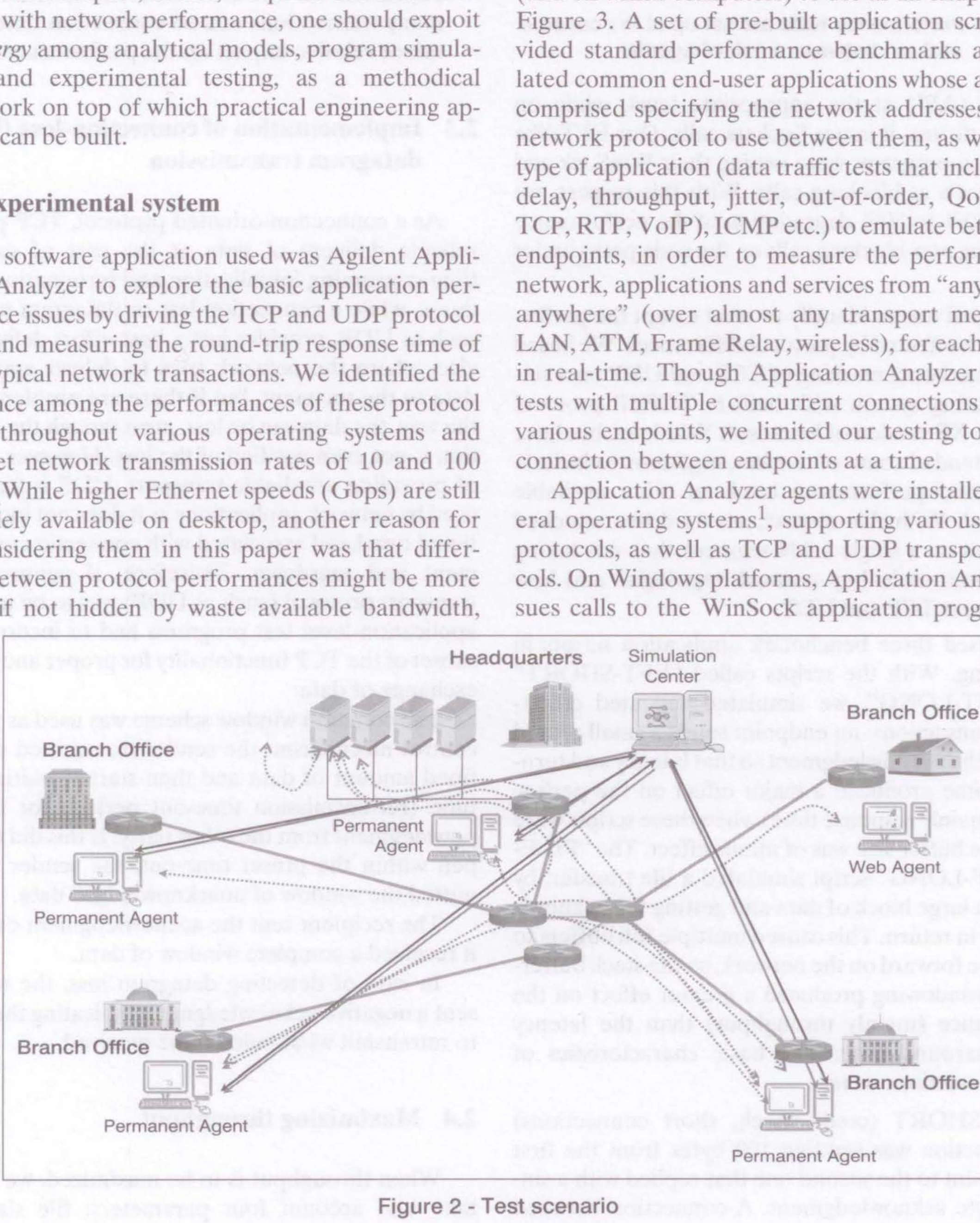

Figure 2 - Test scenario

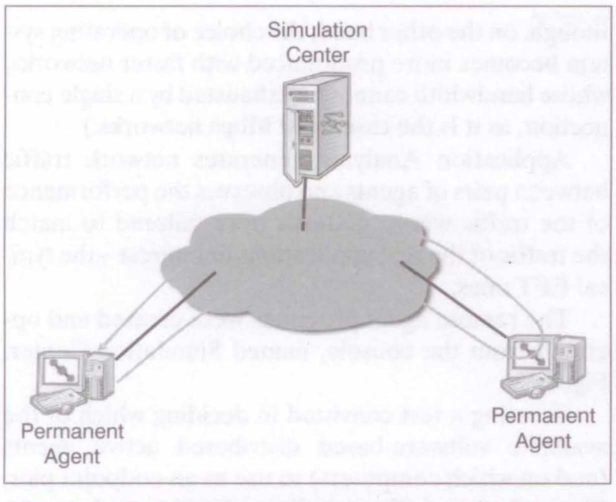

Figure 3 - Performance tests are set up at the console and run between a pair of agents

interface (API) at the appropriate level, while on other platforms, it issues Sockets calls. Our EFT-like application programs were issuing their WinSock and Sockets calls as blocking calls. With this respect, we noticed 20% to 50% degradation of the performance when using non-blocking calls on the code paths under test.

Our goal was to identify to what extent the performance was affected by protocol differences. We based most of our benchmarking of TCP and UDP on a single operating system with built-in TCP/IP protocol stack: the XP version of Microsoft Windows, but later on we extended some of our investigations to include the achieved performance levels of other available platforms[1]. With this respect, we used two matched computers on a single LAN segment thus not taking into account multi-hop network topologies and how routers treat TCP and UDP.

We used three benchmark application scripts in our testing. With the scripts called "EFT-SHORT" and "EFT-LONG", we simulated repeated credit-check transactions: an endpoint sends a small record and gets the acknowledgment so that latency and turnaround time produced a major effect on the performance (mainly response time) when these scripts were run, while buffer size was of minor effect. The "FILETRANSF-LONG" script simulated a file transfer, by sending a large block of data and getting the acknowledgment in return. This caused multiple full buffers to propagate forward on the network, so the stack buffering and windowing produced a greater effect on the performance (mainly throughput) than the latency and turnaround time. The basic characteristics of these transactions were:

- EFT-SHORT (credit-check, short connections) transaction was sending 100 bytes from the first endpoint to the second one that replied with a single-byte acknowledgment. A connection between the endpoints was brought up and torn down within each repeated transaction; the connection time was measured as part of each transaction.
- EFT-LONG (credit-check, long connection) transaction, too, was sending 100 bytes and one byte was received as reply. A single connection was brought up, followed by multiple transactions, before tearing the connection down. Comparing the EFT-LONG to EFT-SHORT reveals the effect of establishing and closing down the connections.
- FILETRANSF-LONG (file-send, long connection) transaction consisted of sending a large number of bytes and receiving a single-byte acknowledgement. As with EFT-LONG, connections span many transactions and so connection time was of minor relative impact on the performance.

2.3 Implementation of connection-less (UDP) datagram transmission

As a connection-oriented protocol, TCP provides reliable delivery of data at the cost of executing time-consuming initialization and termination procedures, while a connection-less or datagram protocol, such as UDP, provides just a best-effort delivery service, where the network tries to deliver application data to the recipient, but if there are problems along the way, the data can be lost, even though the application is not even notified of the loss. However, in spite of providing unreliable transport, UDP is frequently used by network applications as it does not incur additional overhead associated with connection establishment and teardown. Therefore, if connectionless transport protocol (such as UDP) was to be used, our application-level test programs had to incorporate a subset of the TCP functionality for proper and reliable exchange of data:

The common window scheme was used as the flow control mechanism: the sender transmitted a predefined amount of data and then started waiting some time (retransmission time-out period) for the acknowledgment from the other party. If this did not happen within the preset time-out, the sender retransmitted the window of unacknowledged data.

The recipient sent the acknowledgment each time it received a complete window of data.

In case of detecting datagram loss, the recipient sent a negative acknowledgment indicating the sender to retransmit what failed to be received.

2.4 Maximizing throughput

When throughput is to be maximized, we need to take into account four parameters: file size, send buffer size, window size, and maximum transmission unit (MTU). In our benchmark tests between TCP and UDP, we chose such values of these parameters that would maximize the throughput.

File size determines how much user data is to be sent from one program to another. It needs to be large enough to allow accurate measurement of the network, as too small file size yields unrealistically low measure of the network throughput.

Send buffer size is the number of bytes of user data provided on each TCP Send call. It is well-known that IP protocol supports fragmentation and reassembly of datagrams exceeding the packet size. If the TCP/IP network protocol stack can reassemble datagram fragments faster than the application software can issue API Send calls, tests can run faster if we configure the send buffer size as large as possible. On most operating systems, a Send call involves crossing from user space to kernel space; so the fewer API crossings, the better. Once the data make this crossing, they can be sent from the kernel very efficiently. On the other hand, a large number of datagram fragments may increase network congestion and, therefore, the probability that one of them may be dropped. If that occurs, the entire datagram must be retransmitted, causing degradation of the performance.

It is well known that, for Ethernet, the MTU is normally already set at 1,500 bytes, meaning that it is the total data that can be sent on the link, which includes user data and protocol headers (20 bytes for TCP and 20 bytes for IP), so that the amount of user data that can be sent is thus 1,460 bytes, as a single sent block of 32 Kbytes will result in 25 MTUs.

TCP avoids IP datagram fragmentation whenever possible, by breaking data into MTU-sized pieces. Since TCP ensures the delivery of every IP datagram it sends, if one datagram is lost it only requires the retransmission of that datagram. On the other hand, UDP does not avoid IP datagram fragmentation and, whatever size buffer it gets, UDP will pass on the datagram, and then IP fragments it and sends out. So, in this respect, if a 32 Kbyte send data block is issued to TCP, it is broken up by TCP into MTU-sized pieces. If a single MTU is lost, only that MTU needs to be retransmitted. If a 32 Kbyte send data block is issued to UDP, the whole block is passed directly to IP, which breaks it into MTU-sized pieces. If any MTU is lost, the whole datagram is considered lost and the entire 32 Kbyte data block needs to be retransmitted.

Window Size is the amount of user data that can be sent (and the TCP stack must wait) before the acknowledgment is required. A common default for this parameter is 8,760 bytes.

3. TEST RESULTS

3.1 Benchmarking TCP and UDP throughput

The script named FILETRANSF-LONG was used to compare the throughput values achieved with UDP and TCP. Each test used a send file size of 1,460,000 bytes and a send buffer size of 32,767 bytes for TCP, and 8,863 bytes for UDP. We set TCP window size at 8,760 bytes, and the UDP window size at 17,726. With the transmission rate of 100 Mbps, we could set the send file size to 32,543 and the window size to 130,172. The MTU size was left at the maximum of 1,500 for all the tests.

For achieving the best performance, the TCP send buffer size should be as large as possible. Conducting UDP throughput tests requires that the send buffer size is optimized for the actual network. We found the send buffer size of 32,543 bytes to be the best fit for 100 Mbps Ethernet. This did cause IP fragmentation, but not to the extent that it resulted in lost packets and thus degrade the performance. The test results show the difference between UDP and TCP performance, Figure 4. As it can be seen, TCP outperforms the highly optimized UDP (of significantly improved performance), by approximately 2 Mbps.

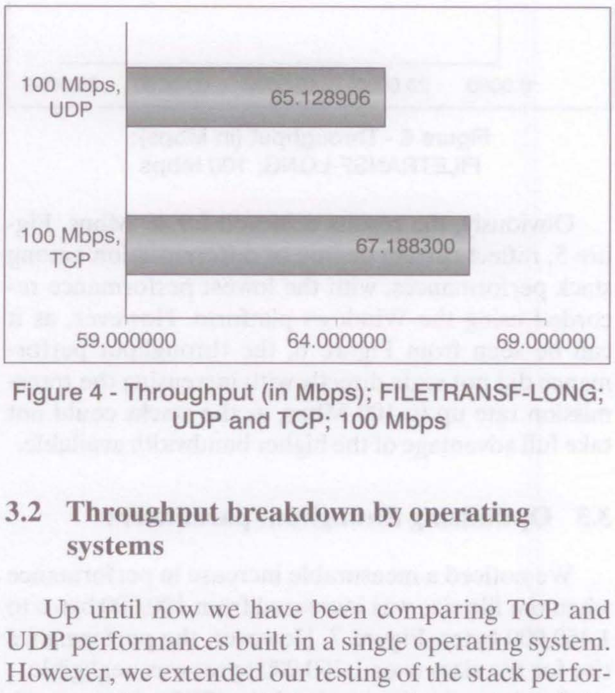

Figure 4 - Throughput (in Mbps); FILETRANSF-LONG; UDP and TCP; 100 Mbps

3.2 Throughput breakdown by operating systems

Up until now we have been comparing TCP and UDP performances built in a single operating system. However, we extended our testing of the stack performance onto other available operating systems (with their shipped stack default parameters). First, we tested throughput, using the FILETRANSF-LONG script, keeping the same operating system for both endpoint agent pairs (except for the Linux tests where we used a Windows computer as the first endpoint). We ran the same set of tests on both Ethernet hierarchy levels.

During each test, 1,460,000 bytes were sent 100 times and each such transfer was timed.

The results are presented in Figures 5 and 6.

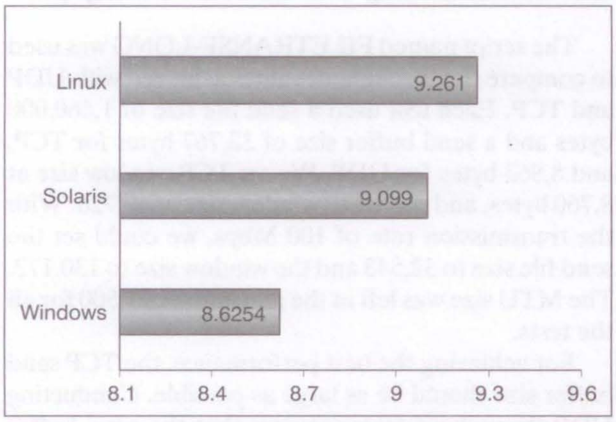

Figure 5 - Throughput (in Mbps); FILETRANSF-LONG; 10 Mbps

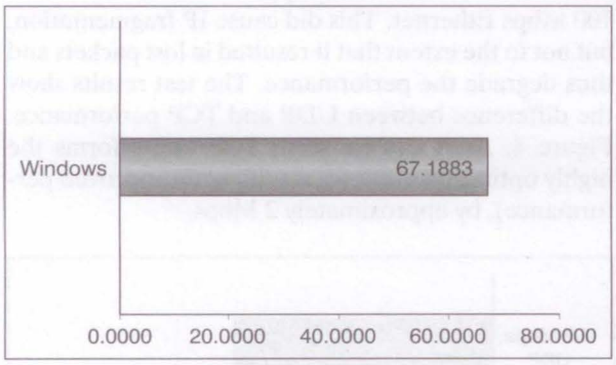

Figure 6 - Throughput (in Mbps); FILETRANSF-LONG; 100 Mbps

Obviously, the results achieved for 10 Mbps, Figure 5, reflect certain degree of differentiation among stack performances, with the lowest performance recorded using the Windows platform. However, as it can be seen from Figure 6, the throughput performance did not scale directly with increasing the transmission rate up to 100 Mbps, as the stacks could not take full advantage of the higher bandwidth available.

3.3 Optimizing throughput parameters

We noticed a measurable increase in performance when the file size was increased from 100,000 bytes to 1,460,000 bytes, Figure 7. However, the performance rise for file sizes over 1,460,000 bytes was negligible.

Our intention was to load the TCP with as much data as possible. Therefore, we set up 32Kbyte for the send buffer size in our tests (though some stacks allow even 64Kbyte). To improve throughput, the TCP stack should send full frames; accordingly, in our tests, multiples of 1,460 bytes were sent. We used a file size of 1,460,000 bytes (with FILETRANSF-LONG), so each sent frame was completely full.

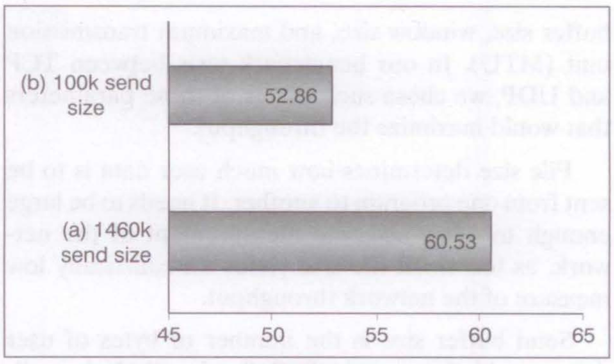

Figure 7 - Throughput (in Mbps); FILETRANSF-LONG: 100 Mbps

To investigate the effect of partially loaded frames, we ran two tests; the first one with the send buffer size of 1,460 bytes (full frame), and the second one with 1,461 bytes. As it can be seen from Figure 8, the performance between the two target endpoint Windows computers, dropped by over 1 Mbps on 10 Mbps Ethernet link.

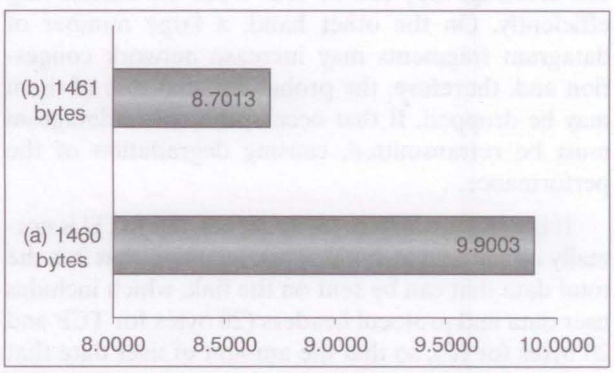

Figure 8 - Throughput (in Mbps); FILETRANSF-LONG; send buffer 1,460 and 1,461 bytes; 10 Mbps

To increase the performance, it is important that network managers understand the network behaviour and know how to optimize network parameters. So, for achieving higher throughput, the number of acknowledgments should be kept low, and, consequently, the window size can be raised close to 64 Kbyte (under condition that all computers can handle it). However, with our hardware, enlarging the window size did not noticeably impact the performance, after we had adjusted other affecting parameters. However, with different network configurations, such e.g. with multiple simultaneous file transfers, the window size is expected to have a greater effect.

3.4 Benchmarking TCP and UDP response time

The response time for both TCP and UDP was tested between a selected pair of computers, connected with 100 Mbps Ethernet. As it can be seen from

Figure 9 and Figure 10, the (connection-oriented) TCP clearly exhibited longer response time, specifically for EFT-SHORT type of transactions (where the connection was opened and closed for each short transaction) with respect to EFT-LONG type of transactions, and regardless of the actual network type. Obviously, the connection setup overhead caused longer response time for short-lasting transactions. The response time could be improved by using long-lasting transactions.

As expected, the response time for UDP was almost equal when using either EFT-SHORT or EFT-LONG scripts, since no connection setup overhead was incurred.

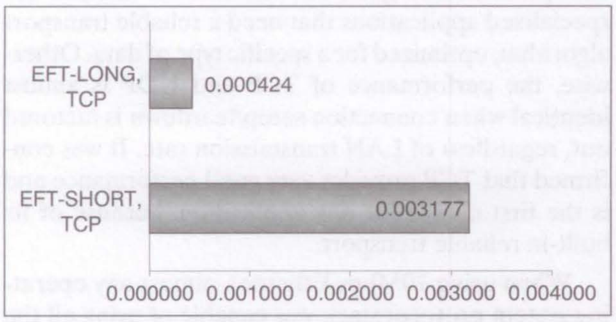

Figure 9 - Response time (in seconds per transaction); EFT-SHORT and EFT-LONG; TCP; 100 Mbps

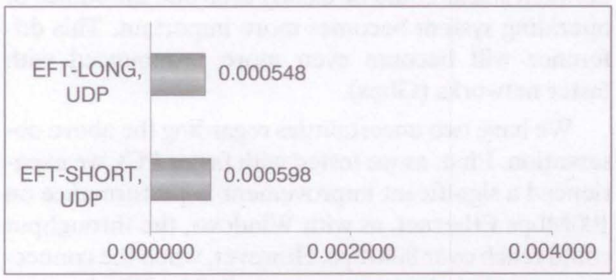

Figure 10 - Response time (in seconds per transaction); EFT-SHORT and EFT-LONG; UDP; 100 Mbps

Testing with EFT-LONG script did not include connection setup overhead, so the benchmarking between TCP and UDP was based just on transmitted data alone. As it is zoomed in Figure 11, better response time was achieved by using TCP's reliable transport algorithm that outperformed the (also reliable) transport algorithm we implemented for UDP. So, when the impact of connection setup and teardown overhead was removed, this test shows that TCP provides an efficient mechanism for achieving excellent response time.

The above results were in accordance with our expectations, as if a selected application was repeatedly sending the same traffic pattern, the earlier described reliable datagram transmission algorithms could be accordingly tuned, so that UDP had solid ground to

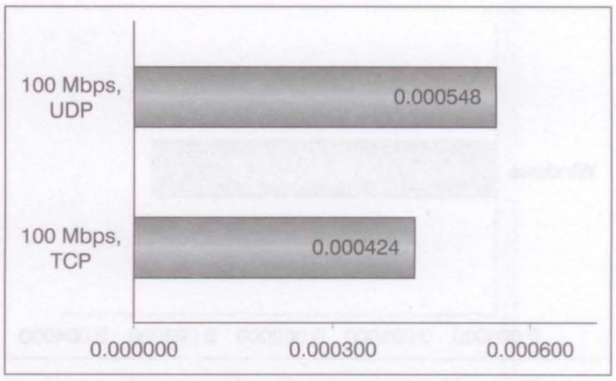

Figure 11 - Response time (in seconds per transaction); EFT-LONG; UDP and TCP; 100 Mbps

outperform TCP. However, as soon as the traffic flow pattern on UDP became more diversified, it became hard to outperform the reliable transport mechanism implemented by TCP. Therefore, we could have justifiably anticipated that TCP performed as well as UDP, or better, in many of such tests of ours.

Furthermore, as TCP provides reliable transport, while UDP allows for low-overhead "stateless" transactions, consequently, the latter performs best with applications that use short transactions. On the other hand, for long-running transactions, TCP is confirmed to be more efficient and able to overcome its inherent connection overhead.

3.5 Response time breakdown by operating systems

The next set of our tests was designed to identify differences in response time among the stacks, where response time is the average time it takes a transaction to be completed. We used the EFT-SHORT script (short transactions which include the connection setup and teardown) in the same configuration as with the preceding throughput tests.

As can be seen from Figures 12 and 13, among all the three targeted operating systems, Windows were found to have the longest response time. To deter-

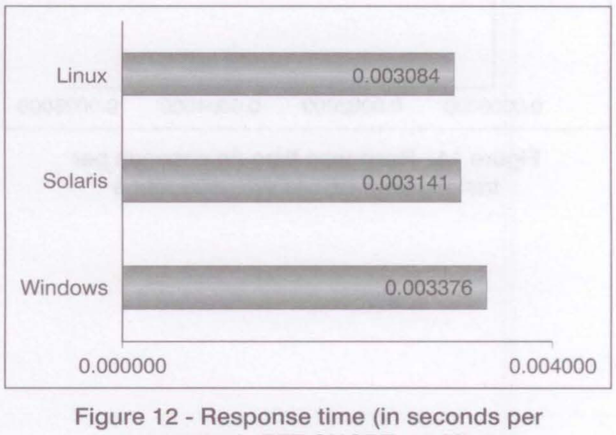

Figure 12 - Response time (in seconds per transaction); EFT-SHORT; 10 Mbps

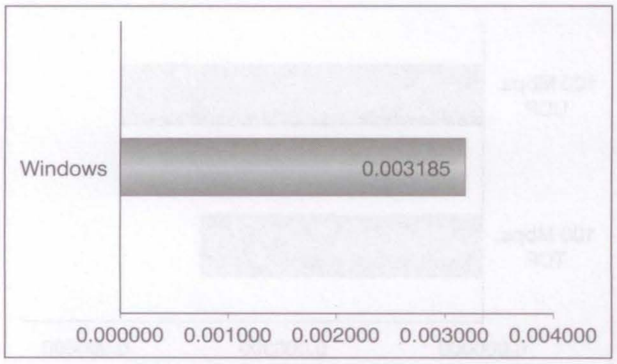

Figure 13 - Response time (in seconds per transaction); EFT-SHORT; 100 Mbps

mine the cause, we closely analyzed the Windows platform, first by running 10 concurrent pairs of EFT-SHORT scripts, expecting to get faster average response, because of the Windows ability to handle multiple tasks. However, the obtained results were the same as with the single pair of agents. Next, we analyzed the line trace of Windows, using the EFT-SHORT script. The line trace showed that the connection setup processing for Windows was consistently faster in this case, as it took between 2 and 3 milliseconds. To validate this observation, we ran the EFT-LONG script between three PC operating systems. As we already mentioned, the EFT-LONG script establishes a connection once and then sends and receives in the same manner as EFT-SHORT. Consequently, running the test with EFT-LONG enabled benchmarking the data transfers without taking into account the connection overhead. As it can be seen in Figure 14, EFT-LONG performance actually improved on Windows. However, the connection

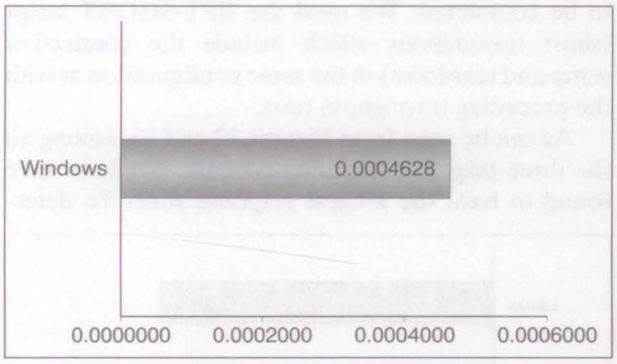

Figure 14 - Response time (in seconds per transaction); EFT-LONG; 100 Mbps

setup was slower on Windows than on the other two PC operating systems.

4. CONCLUSION

It has been demonstrated that TCP provides high level of performance and should be the protocol of choice with regard to UDP, for most applications. The TCP performance is acceptable whenever the time for connection setup and teardown is not relatively significant, such as with file transfers. If an application needs to avoid connection overhead, UDP can provide performance gain, but can also be valuable for specialized applications that need a reliable transport algorithm, optimized for a specific type of data. Otherwise, the performance of TCP and UDP is almost identical when connection setup/teardown is factored out, regardless of LAN transmission rate. It was confirmed that TCP provides very good performance and is the first choice for any application, because of its built-in reliable transport.

When using 10Mbps Ethernet, almost any operating system protocol stack was capable of using all the bandwidth with just one connection, but with 100Mbps Ethernet links, the performance difference between stacks could be clearly seen and the choice of operating system becomes more important. This difference will become even more pronounced with faster networks (Gbps).

We have two uncertainties regarding the above observation. First, as we tested with faster PCs, we experienced a significant improvement in performance on 100Mbps Ethernet, as with Windows, the throughput could reach over 80Mbps. However, when the connection setup/teardown was not negligible, Windows exhibited the longest response times. This observation remained even when the connection setup/teardown could be ignored.

Therefore, much remains to be further explored in this area, as we have just "scratched the surface" in gaining understanding of how these protocol performances behave in real situations, involving not just a single connection, but a number of concurrent connections, as well as intermediate network devices, such as routers and switches, that also influence aggregate throughput and response time.

Dr. sc. **VLATKO LIPOVAC**
E-mail: vlatko.lipovac@unidu.hr
Dr. sc. **VEDRAN BATOŠ**
E-mail: vedran.batos@unidu.hr
Sveučilište u Dubrovniku
Odjel za elektrotehniku i računarstvo
Branitelja Dubrovnika 29,
20000 Dubrovnik, Republika Hrvatska
Dr. sc. **ANTUN SERTIĆ**
E-mail: antun.sertic@fpz.hr
Sveučilište u Zagrebu, Fakultet prometnih znanosti
Vukelićeva 4, 10000 Zagreb, Republika Hrvatska

SAŽETAK

ISPITIVANJE APLIKACIJSKIH PERFORMANSI MREŽA S EFT PROMETOM

U ovome radu istražujemo kako krajnje performanse (posebno vezano uz promet elektroničkih financijskih transakcija - EFT) ovise o korištenom stogu protokola, operacijskom sustavu i brzini prijenosa mrežom. S obzirom na navedeno, provedeni su odgovarajući simulacijski testovi performansi TCP i UDP protokola, instaliranih na različitim operacijskim sustavima, počevši od sustava Windows, Sun Solaris, do sustava Linux, i uočene su razlike u performansama, fokusirajući se na propusnost i vrijeme odziva.

KLJUČNE RIJEČI

aplikacijske (krajnje) performanse, kvaliteta usluge, EFT promet

REFERENCES

1. Operating systems tested: Windows, Linux and Sun Solaris.

LITERATURE

[1] **Comer, D. E.**, "*Internetworking with TCP/IP*, Volume 1; Principles, Protocols, and Architecture (Fifth Edition), Prentice Hall, NJ, 2005

[2] **Tannenbaum A. S.**, "*Computer Networks* (Fourth Edition), Prentice Hall PTR, NJ, 2002

[3] **Agilent Technologies**, *Application Analyzer Command Line Interface Guide*, Colorado Springs, 2002

[4] **Quinn, B.** and **D. Shute**. *Windows Sockets Network Programming*. Addison-Wesley, Reading, MA, 1995

[5] **Stevens, W. R.** *TCP/IP Illustrated*, Volume 1. Addison-Wesley, Reading, MA, 1994