An adaptive approach to optimize thin client protocols

Frédéric Fok A.C.[1], Benoit Lécroart[1], Émilie Chan[1], Pieter Simoens[2], Bart Dhoedt[2]
[1]*NEC Technologies (UK) Ltd, Immeuble Optima, 10 rue Godefroy, 92821 Puteaux, France.*
Tel: +33 1 5568 8200, Fax: +33 1 5568 8210, Email: {frederic.fok, benoit.lecroart, emilie.chan}@nectech.fr
[2]*INTEC – IBCN, Ghent University – IBBT,*
Gaston Crommenlaan 8 bus 201, 9050 Gent, Belgium
Tel: +329 331 49 00, Email: {pieter.simoens, bart.dhoedt}@intec.ugent.be

Abstract: In a thin client computing environment, applications run on server farms and are accessed through thin client protocols like Remote Frame Buffer (RFB), or Remote Desktop Protocol (RDP). Although they work well in high and stable bandwidth networks, thin client protocols require some modifications to adapt to the varying characteristics of wireless networks and to reduce the energy consumption of thin client mobile devices. In this paper we present both upstream and downstream adaptive protocol optimizations. In the upstream direction, user events are buffered to reduce packetization overhead. In the downstream direction an adaptive scheduling-based transmission pattern is adopted. Our solution shows we can reduce the bandwidth consumption while preserving the Quality of Experience. Those adaptive mechanisms were developed as part of MobiThin FP7-funded European project.

Keywords: Thin Client, Adaptive protocols, wireless networks, Quality of Experience

1. Introduction

In a thin client computing environment, applications run on server farms. Through thin client protocols, the client sends user data such as key entries or mouse movements to the server, which on its turn sends the graphics output data generated by the applications.

The benefits of the thin client computing paradigm are well known: reduction of total cost of ownership, improved security and privacy. Today thin client technology is getting much more interest thanks to virtualization technologies and the ability to take benefits of CPU processing power on remote servers distributed over the network.

Different thin client protocols exist, such as Virtual Network Computing's Remote FrameBuffer Protocol (VNC-RFB) [1], Microsoft's Remote Desktop Protocol (RDP) [2], Citrix' ICA (Independent Computing Architecture) [3], or X11 [4]. These thin client solutions provide rather good Quality of Experience (QoE) in a high and stable bandwidth network environment and are promising in the mobile wireless networks.

These thin client protocols adopt either a polling transmission pattern wherein the client regularly requests the server to send a screen update, or a pushing transmission pattern wherein the server autonomously sends the screen refresh to the client. In the push model, the screen refresh rate can be unnecessary higher than the real user needs, leading to overused bandwidth and higher CPU consumption. In the pull model, the screen refresh is naturally adapted to client needs. In wide area networks, the network round trip time (RTT)

can be very high so the refresh rate can become very low while the network bandwidth is underused. In contrast to local area networks where the RTT is low, the refresh rate can be too fast with respect to the real user needs and device capabilities, which also leads to high CPU and bandwidth consumption. In upstream direction, the user events are sent immediately as they occur over TCP/IP. The large packetization overhead, due to transmitting every user event in its own TCP/IP packet and the TCP acknowledgments (ACK), result in sub-optimal bandwidth consumption. As such, current thin client protocols working over TCP protocol consume a lot of bandwidth and the QoE is also reduced because of the TCP retransmission effects in wireless network [5]. Furthermore in mobile wireless environment, the existing thin client protocols have to face strong network impairments (handover, fluctuation of radio signals, bandwidth, packet loss or delay).

All those elements show that it is possible to optimize the screen update transmission and user event transmission by taking into account the network conditions, the user needs, the client and the server capabilities. MobiThin is a European FP7-funded project that is designing and developing a solution that will enable intelligent distribution of demanding services and applications to mobile thin client devices [6]. In addition to the design of a suitable framework for thin client service delivery and provisioning, research activities are made into thin client protocols to adapt to wireless network characteristics.

The remainder of this article is structured as follows. Section 2 presents the system architecture and the main blocks of the solution involved in the end-to-end protocol adaptation. In section 3 we describe the adaptive mechanisms for both the downstream and upstream directions. The experimental results are finally discussed in Section 4.

2. Overall architecture

The mobile thin client solution will work in both current internet and mobile wireless network infrastructure. Such architecture has been designed by the MobiThin project [6]. The thin client protocol should adapt to the varying network conditions and be efficient in term of power and bandwidth consumption.

In the field of downstream thin client protocol traffic, several studies were conducted to evaluate the performance of thin client protocols. [7] discusses the performance of remote display mechanisms, adopting server-push or client-pull models, across broad range of systems. In addition, to cope with reduced bandwidth consumption on the downlink direction, various compression techniques can be used. However none of the studies known so far considered remote display mechanisms that adapt to the overall system conditions. This reflects the need for a thin client protocol coping with dynamic variations of the environment such as handovers, delay variation, decreasing level of mobile device battery, user expected QoE, variation of server load and capabilities. The protocol will also have to ensure sufficient QoE in a mobile wireless environment.

To our knowledge, related work in the field of upstream thin client protocol traffic is currently limited to traffic characterization, a.o. by the University of Wurzburg [8][9]. For the Independent Computing Architecture (ICA), Citrix' thin client protocol, the authors demonstrate how nearly all packets sent by the client are either TCP acknowledgements or IP packets containing a single user event. Furthermore, they point out that in the design of the ICA protocol, responsiveness has played a more important role than packetization overhead. In [10], we have quantified the power consumption of the wireless platform on thin client devices due to thin client protocol traffic. Although the WNIC is mostly in idle state, the results indicate that transmitting and receiving data account for up to 30 % of the total WNIC power consumption, even under circumstances of low packet loss and only medium congestion. This illustrates the need to optimize thin client protocol bandwidth consumption.

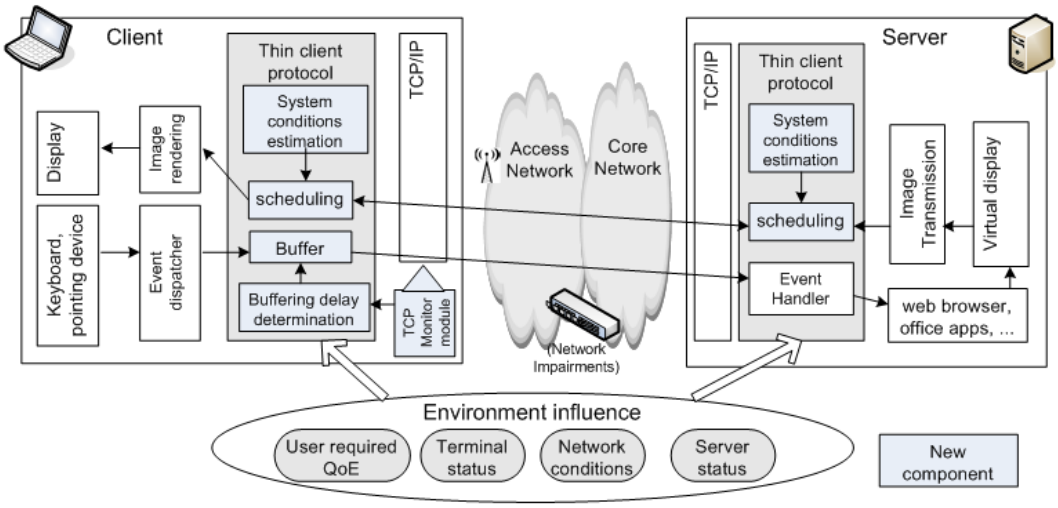


Figure 1: Thin client protocol adaptation to environment constraints

In this paper, as illustrated in Figure 1, we propose two protocol optimizations. In the downstream direction, a scheduled-based screen update transmission pattern is introduced, where the scheduled refresh is adapted to the client needs and current wireless network conditions. In the upstream direction, a user event buffering mechanism is presented that decreases the packetization overhead while taking into account user responsiveness.

3. Protocol adaptivity mechanisms

3.1 Protocol adaptivity for downstream direction

We propose a transmission mechanism driven by the idea that the screen updates are scheduled based on terminal, server and network conditions, as well as user expectations.

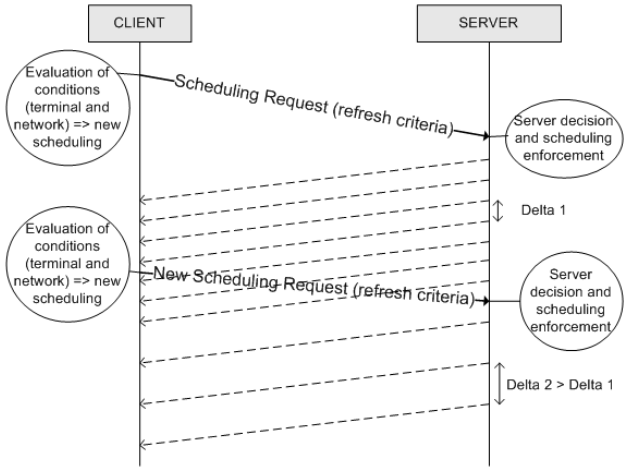


Figure 2: Scheduling pattern for adaptive screen updates

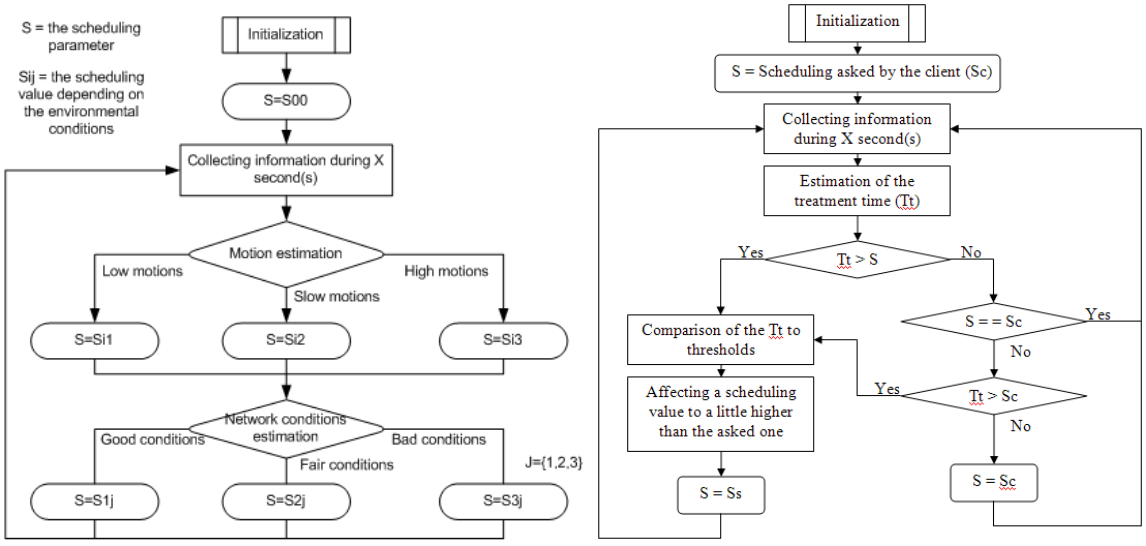
As shown in Figure 2, based on system conditions (battery level, processing speed, supported decoding, available bandwidth, delay, etc.) the client estimates a scheduling value which it sends to the server in the form of a “Scheduling Request” message. Besides the maximum update frequency, this message can contain specific refresh criteria (such as the area of interest and the desired encoding scheme). Upon receipt of the Scheduling Request message the server may re-estimate the scheduling value before enforcing the scheduling onto the screen update transmission. Therefore, as soon as the server detects some areas having been updated the server transmits the screen data to client based on the

predefined scheduled value. This process will be regularly applied leading to a dynamic adaptation of the thin client protocol.

By using this pattern to dynamically adapt to the client and server capabilities and the varying network conditions, we expect to minimize the power consumption by adapting the refresh rate to the real user needs and to the device capabilities as well as optimally use the available bandwidth. The adaptive scheduling algorithm can also prevent slowdowns caused by end-to-end congestion by slightly increasing the scheduled value to let the client the time to render the screen updates. Therefore better bandwidth consumption with acceptable QoE, even on degraded network conditions, is expected for some use cases (office application, internet browsing). Although the pattern may prove to be advantageous when the user plays a video we do not expect much in this area and believe other transport protocol and encoding means are more suitable, for example presented in [11].

The proposed pattern was implemented on TightVNC [12] that is based on an optimized version of the Remote Frame Buffer protocol.

3.1.1 Client and server scheduling algorithms



(a) Client side algorithm ***(b)*** *server side algorithm*

Figure 3: scheduling value evaluation

As shown in Figure 3(a), the client estimates the scheduling value based on motion and network conditions estimation. The motion estimation was implemented by calculating the percentage of pixels to update per frame. The system conditions were estimated by calculating the display time and updates' receipt frequency. The system conditions are network impairments such as bandwidth reduction, client impairment such as decoding capability, or server impairment such as reduced transmission rate.

The display time is the time elapsed between the receipt of the first pixels received in a framebuffer update and the display of the last pixels of the same framebuffer update on the client's screen. If this time reaches a certain threshold we conclude there is bandwidth restriction because the data takes longer time to be transmitted, or there is client congestion because it takes more time to treat the received data.

The updates' receipt frequency is an average of the measured real scheduling calculated by the client. This measure represents the time between the receipts of two frame updates. If the calculated value is above a certain threshold we can conclude there is client or server congestion, or bad network condition.

The server, as shown in Figure 3 (b), also evaluates the average scheduling value so it can estimate the congestion. When this mean scheduling value is too long this is probably due to server congestion, client congestion or a bandwidth restriction. In our implementation, when congestion is detected, the server applies a scheduling higher than the one asked by the client to let the client the time to treat all pending updates thereby reducing the congestion. If this dynamic adaptation was not activated, the client could not be able to display these update properly and in this case the user may suffer from a sensation of slowdown.

3.2 Protocol Adaptivity for Upstream direction

In general, three types of messages dominate the upstream traffic generated by pull thin client protocols. A first source are the acknowledgements (ACK) that are generated by TCP to ensure reliable transmission. The other two types are generated by the thin client protocol itself, namely user events and requests for display updates.

These observations are confirmed by Figure 4 depicting a histogram of the IP packet size, captured in the upstream direction from client to server during a typical VNC session involving text editing and browsing. Table I gives an overview of the size of the VNC-RFB protocol message.

Typically, user input is encoded as a series of small packets generated quickly one after another, resulting in a major packetization overhead. For example, moving the pointer results in a series of PointerEvent messages that only contain the new coordinates. A key stroke results in two KeyEvent messages, one for the press and one for the release action.

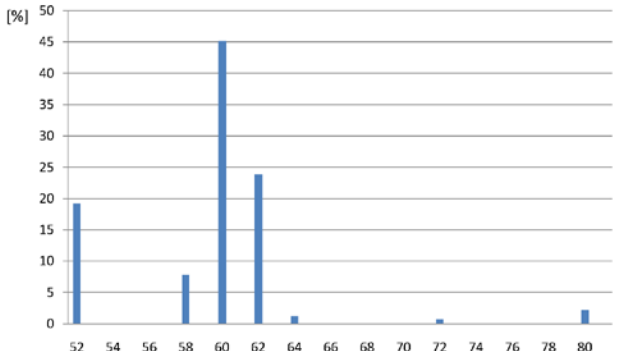


Figure 4: Histogram of packet size of upstream VNC traffic

Table 1: Packetization overhead of upstream RFB protocol messages

RFB message	data size [B] RFB	IP	overhead [%]
FBUpdateRequest	10	62	83.87
KeyEvent	8	60	86.67
PointerEvent	6	58	89.66

Every single user event is encapsulated in a separate TCP/IP packet, which results in a major packetization overhead. KeyEvents contain 8 bytes of data, but are prepended with a 32-byte TCP header and a 20-byte IP header, accounting for a packetization overhead of 87%. Similarly, the TCP/IP packetization of PointerEvents leads to an overhead of 90%.

3.2.1 Adaptive user event buffering

A major fraction of the packetization overhead could be reduced if a single TCP segment would contain multiple VNC-RFB protocol messages. This can be achieved by buffering the user events at the VNC layer. However, this increases the user perceived latency between the user input and the result becoming visible on the screen. Therefore, the buffering time should be adapted to the current network roundtrip time, in order not to harm the user interactivity experience.

A straightforward approach to determine the buffering delay BD(t) at time t can be proposed as:

$$BD(t) = UPPER\_BOUNDARY - RTT(t) \quad (1)$$

The value for the UPPER_BOUNDARY can be adapted according to the requirements of the application at hand. As indicated in [13], users tolerate less delay for highly interactive applications. While for text editing a delay is tolerated of up to 150 ms, for gaming mostly delays up to 80 ms are accepted.

3.2.2 Implementation

On a Linux platform, we have extended a TightVNC server with the adaptive buffering algorithm, as shown in Figure 1. A buffer was implemented between the RFB protocol layer and the TCP layer. The TCP layer was extended with our own kernel module to expose TCP state information to the other layers. More specifically, this module was used to retrieve the network roundtrip time estimation that is maintained by the TCP protocol. Based on this information, the optimal buffering delay is configured.

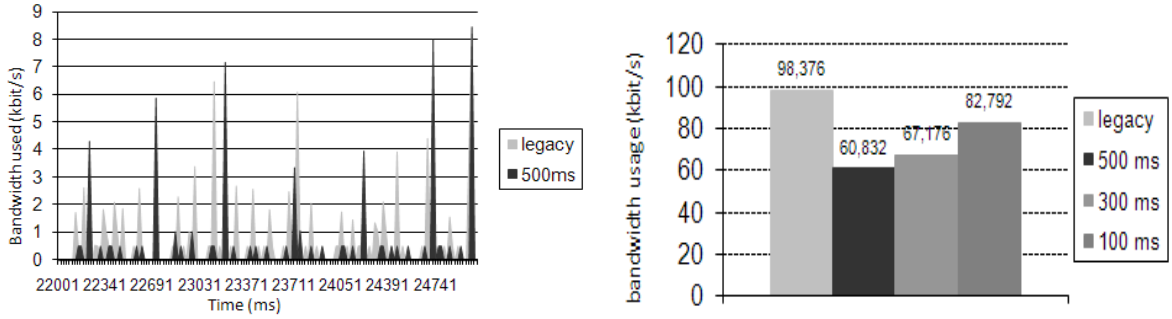
4. Performance evaluation

4.1 Downstream Experimental Results

To assess the improvements of our optimizations, we have set a test bed wherein a gateway was used to shape the bandwidth. The tests consisted of comparing the bandwidth used by the legacy client with the enhanced one for different scenarios, and with different scheduling periods. All user events were recorded and automatically executed to guarantee reproducibility of the scenarios.

4.1.1 Results in high bandwidth network (LAN)

Figure 5 (a) presents the bandwidth used by the VNC client while the user is writing a text in an office text editing application with a refreshment period of 500 ms. The bandwidth comprises the upstream and downstream data. This rather high scheduled period value allows making more visible the differences comparing the enhanced protocol from the legacy protocol.



(a) *Comparison of Bandwidth usage* ***(b)*** *Cumulated bandwidth usage during 3,282s*

Figure 5: bandwidth consumption, for a text application, in high available bandwidth network

While the legacy client receives several little updates, the enhanced client receives less but bigger updates regrouping all the modifications made since the previous update received. If we look at the cumulated bandwidth values, on the Figure 5 (b), we can see that a connection with a legacy client always uses more bandwidth than with the enhanced client. In this scenario, the higher the scheduled period value the lower the overall bandwidth consumed. This represents a bandwidth gain from 15 to 40%.

To balance the bandwidth consumption with good QoE, the user can determine which level of interactivity he finds acceptable. Besides, previous studies suggest round trip delays from 150ms to 1s depending on the interactivity requirements [13]. From experience we can conclude that a scheduling value of 300ms is a good trade-off.

The proposed pattern presents some benefits since a modification of the scheduled period can reduce the bandwidth consumption while preserving same or slightly lower QoE. Another benefit is that the pattern will naturally reduce the radio activity that is an important factor for a mobile terminal to increase the device autonomy.

4.1.2 Results in low bandwidth network

In Figure 6, we have limited the bandwidth at 15kbit/s. We compared a legacy viewer with the modified viewer with a scheduled updated period set to 300ms.

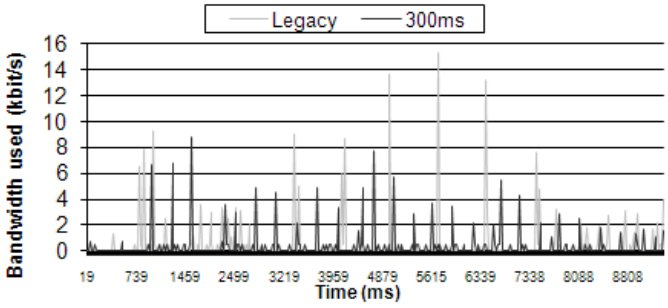


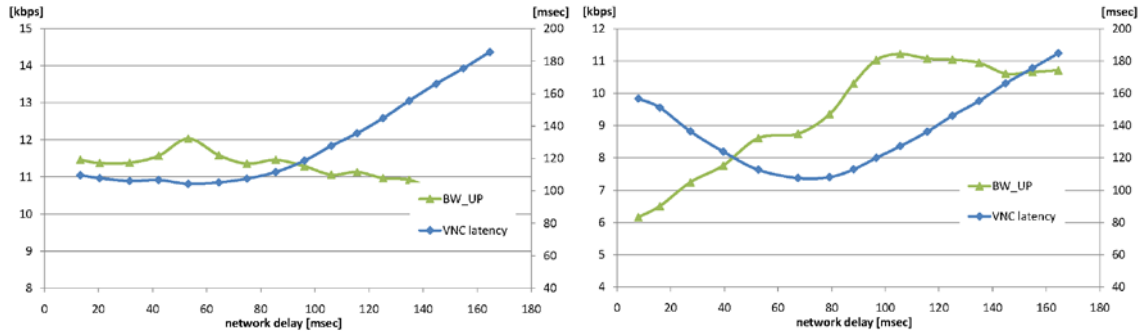
Figure 6: bandwidth used for a text application with a limitation of 15kbit/s

We can observe that the modified viewer is not much affected by this limitation. Contrary to the legacy client, the updates of the modified viewer are regular: the quality of experience is still good for client with a scheduled value of 300ms while the legacy client's QoE is worse. If the limitation is reduced to 10kbit/s, the modified viewer starts to give lagged results and for a better quality of service, the client has to increase its scheduled period. A 600ms-client seems to be an interesting trade-off in those conditions, giving a slightly improved QoE over the legacy viewer.

4.2 Upstream Experimental Results

For the experimental validation, the reference architecture depicted in Figure 1 was used. The buffering algorithm was evaluated for increasing values of network delay up to 150 ms and each experimental run comprised 20 iterations, in order to factor out random noise. The VNC latency is defined as the average time between the client sending a FramebufferUpdateRequest and receiving the subsequent FramebufferUpdate.

For reference purposes, the behaviour of unmodified VNC under increasing network delay is presented in Figure 7 (a). Without user event buffering, the upstream bandwidth slightly decreases for higher network delays, due to longer TCP roundtrip delays. In Figure 7 (b), the user event buffering is applied, with an UPPER_BOUNDARY of 100 ms. Referring to Equation (1), for network delays below 100 ms, the user event buffering algorithm can be applied and hence the bandwidth is drastically reduced up to 6 kbps for a RTT of 80 ms. The fact that the latency is higher than the UPPER_BOUNDARY is due to processing delays at the server. In some cases, the server does not respond immediately to a FramebufferUpdateRequest, because the screen has not changed since the previous screen update, and a deferred update mechanism is activated [1]. Typically, screen changes result from user input, e.g. in a text editor and buffering user events will lead to more deferred updates. This increases the total perceived latency. In a next version of the algorithm, we will take into account the server processing.



(a) Unmodified *(b) UPPER_BOUNDARY = 100 ms*

Figure 7: VNC latency and upstream bandwidth for different values of the network roundtrip time, with and without user event buffering.

5. Conclusions

In this paper, adaptive thin client protocol optimisations were presented for both the upstream and the downstream direction. In the downstream direction, an adaptive and scheduled-based screen update transmission pattern was introduced. We highlighted the benefits of reduced bandwidth and energy consumption with still a good or acceptable QoE. In the upstream direction, the buffering of user events decreases the packetization overhead and leads to important bandwidth savings. The buffering period is adapted to the current network delay, in order to preserve the user perceived latency.

The adaptive protocol, combined with ongoing MobiThin research activities, is expected to ease the introduction of thin client applications into mobile wireless network using mobile terminals.

Acknowledgments

Part of the research leading to these results was done for the MobiThin project and has received funding from the European Community's Seventh Framework (FP7/2007-2013) under grant agreement nr 216946.

References

[1] T. Richardson, Q. Stafford-Fraser, Kenneth R. Wood, and Andy Hopper, *Virtual Network Computing*, IEEE Internet Computing, 1998, Vol.2, Issue 1, pp. 33-38.

[2] Microsoft, Windows Remote Destkop Protocol [RDP] and Windows Terminal Services, http://www.microsoft.com

[3] C. Inc., Citrix Independent Computing Architecture (ICA) and Citrix Xen App, http://www.citrix.com

[4] X.Org Foundation project, an open source implementation of the X Window System, http://www.x.org

[5] Hari Balakrishnan, Venkata N. Padmanabhan, Srinivasan Seshan, and Randy H. Katz. A comparison of mechanisms for improving TCP performance of wireless links. *IEEE/ACM Transactions on Networking*, 6(5), December 1997.

[6] MobiThin, http://www.mobithin.eu

[7] S. Jae Yang Nieh, Matt Selsky, and Nikhil Tiwari. The Performance of Remote Display Mechanisms for Thin-Client Computing. USENIX Association, Montery, California, USA, June 10-15, 2002.

[8] Emmert B, Binzenhofer A, Schlosser D, Weiss M. "Source traffic characterization for thin client based office applications". In Lecture Notes In Computer Science 2007; pp 86-94.

[9] Schlosser D, Binzenhofer A, Staehle B. "Performance comparison of Windows-based thin client architectures". In Proceedings of Australasian Telecommunication Networks and Applications Conference (ATNAC) 2007.

[10] Simoens P, Vankeirsbilck B, Ali F.A., Deboosere L, De Turck F, Dhoedt B, Demeester P. "Characterization of power consumption in thin clients due to protocol data transmission over IEEE 802.11".

[11] P.Simoens et al. Design and implementation of a hybrid remote display protocol to optimize multimedia experience on thin clients. Australasian Telecommunication Networks and Applications Conference, December 2008

[12] TightVNC Software, http://www.tightvnc.com

[13] Niraj Tolia, David G. Andersen, M. Satyanarayan. Quantifying Interactive User Experience on Thin Clients. IEEE Computer, Volume 39 – 3, pages 46-52, March 2006.