



**IST-4-027187-STP-SURFACE**

**D7.6 v1.0**

***Performance evaluation of TCP/IP based applications over the Self Configurable Air Interface***

**Contractual date of delivery to the EC: 15<sup>th</sup> January 2009**

**Actual date of delivery to the EC: 21<sup>st</sup> January 2009**

**Editor (s): Troels B. Sørensen (AAU)**

**Author (s): Oumer Teyeb (AAU), Troels B. Sørensen (AAU)**

**Workpackage: WP7**

**Estimated person months: 1**

**Dissemination level<sup>1</sup>: PU**

**Nature: Report**

**Version: 1.0**

**Total number of pages: 19**

**Abstract:**

This report describes preliminary performance results of TCP and application layer protocols over the unified air interface. The results are generated using the real time emulator developed for the SURFACE project. File transfer using the FTP protocol and video streaming via real time streaming server are the applications that have been investigated. The performance of these applications are analysed for a user located at different distances from the base station. Performance results are given using the key performance indicators of user throughput, packet loss and packet delay.

**Key words: real-time emulation, end-to-end service evaluation, FTP, Streaming**

---

<sup>1</sup> PU: Public, PP: Restricted to other programme participants (including the Commission Services), RE: Restricted to a group specified by the consortium (including the Commission Services), CO: Confidential, only for members of the consortium (including the Commission Services)

**DISCLAIMER**

The work associated with this report has been carried out in accordance with the highest technical standards and the SURFACE partners have endeavoured to achieve the degree of accuracy and reliability appropriate to the work in question. However since the partners have no control over the use to which the information contained within the report is to be put by any other party, any other such party shall be deemed to have satisfied itself as to the suitability and reliability of the information in relation to any particular use, purpose or application.

Under no circumstances will any of the partners, their servants, employees or agents accept any liability whatsoever arising out of any error or inaccuracy contained in this report (or any further consolidation, summary, publication or dissemination of the information contained within this report) and/or the connected work and disclaim all liability for any loss, damage, expenses, claims or infringement of third party rights.

**CONTENT LIST**

Definitions, symbols and abbreviations .....	4
1 INTRODUCTION.....	5
2 SIMULATOR-EMULATOR INTERFACE AND MODELLING ISSUES .....	7
3 EMULATOR GUI.....	9
4 PERFORMANCE EVALUATION .....	12
4.1 INVESTIGATED SCENARIOS.....	12
4.2 FTP .....	13
4.3 STREAMING .....	14
5 CONCLUSION .....	18
6 References .....	19

**DEFINITIONS, SYMBOLS AND ABBREVIATIONS**

<b>AM</b>	Acknowledged Mode
<b>ARQ</b>	Automatic Repeat reQuest
<b>BS</b>	Base Station
<b>cdf (C.D.F.)</b>	Cumulative Distribution Function
<b>DL</b>	Down Link
<b>FTP</b>	File Tranfer Protocol
<b>GUI</b>	Graphical User Interface
<b>HARQ</b>	Hybrid ARQ
<b>IP</b>	Internet Protocol
<b>MAC</b>	Medium Access Control
<b>MIMO</b>	Multiple Input Multiple Output
<b>QoS</b>	Quality of Service
<b>RLC</b>	Radio Link Control
<b>RRM</b>	Radio Resource Management
<b>SACK</b>	Selective ACKnowledgement
<b>SC-FDMA</b>	Single Carrier - Frequency Division Multiple Access
<b>SDU</b>	Service Data Unit
<b>TCP</b>	Transmission Control Protocol
<b>TM</b>	Transparent Mode
<b>TTI</b>	Transmission Time Interval
<b>UDP</b>	User Datagram Protocol
<b>UL</b>	Uplink
<b>UM</b>	Unacknowledged Mode
<b>UTOD</b>	Unitary Trace Orthogonal Design
<b>VBR</b>	Variable Bit Rate
<b>VLC</b>	VideoLAN Client

## 1 INTRODUCTION

Quality of service (QoS) is usually identified by basic objective performance metrics such as delay, throughput and jitter. However, it is also important to investigate the end user's subjective perception as the main impact of service quality is on the end user. Using a real time emulation platform, it is possible to do both objective and subjective investigations.

As specified in [1] and [2], an air interface emulation platform has been developed to assist in the evaluation of the end-user experience over the proposed SURFACE air interface. The emulator captures packets from real world applications and, by delaying and/or dropping them (according to the scenario under investigation), is able to reproduce the overall impact of the air interface and different protocols on end-to-end quality of service.

The usage scenario of the emulator, as it has been designed, is shown in Figure 1. It consists of an emulation and application server, the latter providing services such as FTP, web browsing and streaming. The emulation server, which also acts as a proxy for accessing the application server, intercepts IP packets from a connected client, in this case connected via the internet as illustrated on the left, and subjects these to the conditions on the SURFACE uplink air interface. The scenario being emulated is thus a client on the fixed network accessing a mobile application server. With this usage scenario, the client user will be able to experience the performance of different realtime and non realtime services delivered from the (mobile) application server through the emulator.

Different trace files are generated from system-level simulation of the SURFACE uplink air interface, under different system settings, and the emulator uses these files to manipulate the packets that it is intercepting. Actions taken by the emulator (which are basically delaying and dropping packets) will be recorded in log files for later analysis. Thus, both user level/subjective and statistical/objective analysis of performance are possible.

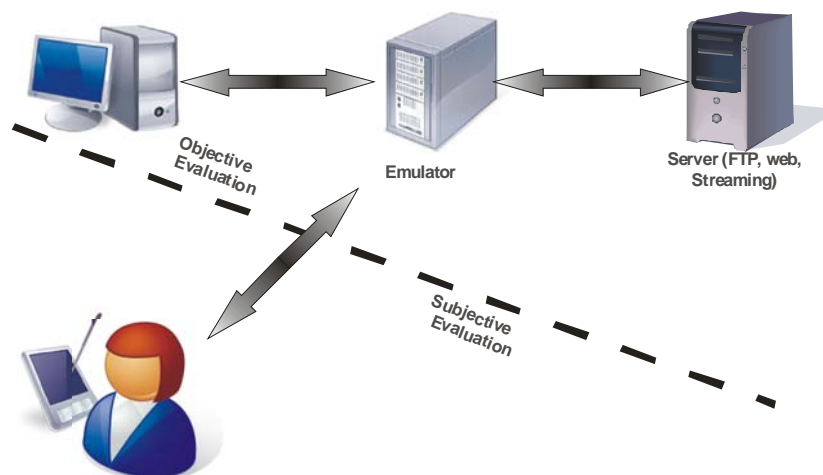


Figure 1: Emulator usage scenario (taken from [2]).

This deliverable describes preliminary performance results of the unified air interface evaluated through the SURFACE emulation platform. Chapter 2 describes how the emulation is coupled with system-level simulation traces. In Chapter 3, the graphical user interface (GUI) that can be used to setup the emulator as well as to get real time feedback on the performance is described. Performance results for FTP and streaming services are then given in Chapter 4. Finally, a brief conclusion is given that summarizes the main points of this deliverable.

## 2 SIMULATOR-EMULATOR INTERFACE AND MODELLING ISSUES

A proper interface between system-level simulations and the emulator is required so that the emulation process will match the behaviour of the SURFACE air interface as close as possible. In [2], implicit, explicit and emulator-simulator coupling were described as possibilities of this interfacing and the emulator-simulator coupling was recommended as the most practical interfacing method. However, due to further practical limitations of carrying out the emulation and simulation processes in parallel, it has been decided that the simulation and emulation processes were carried out separately. First, system-level simulations are performed and the results are summarized in trace files. Then these traces are fed to the emulator, which will use them to manipulate the packets of the applications that it is intercepting.

The trace files contain the TTI per TTI evolution of the system-level simulation. Table 1 shows sample trace file data to illustrate the trace file format.

**Table 1: Trace file format**

Time stamp	Payload	Status	Delay	Cell Payload
1	324	0	0	234
2	324	0	1	1243
3	324	1	2	5323
4	2331	1	0	5124
...	.....	.....	.....	.....

The first column, Time stamp, is the TTI index (arbitrary offset) at which the transmission of data from the user begins. The TTI duration used during the system-level simulations is set to 0.5 ms. The second column is the user payload, which is the (attempted) throughput in bits during that TTI. A status of 1 indicates success of the transmission carried out during this TTI, while 0 is a failure. Delay is the number of TTIs that it takes to transfer the respective payload with success (excluding the inherent one TTI delay in sending a packet from transmitter to receiver). The last column, cell payload, is the successful throughput in kbits for all users in that cell during the TTI.

Note that the trace file is only for one chosen user during the simulation process, and that only one HARQ process per a given connection is assumed (i.e. when the data contained in a TTI is not received properly, it will be retransmitted in the next TTI). For example, from the trace file of Table 1, we can see that TTI #1 was not received properly, so we have retransmissions at TTI #2 (which is also not correctly received). Finally, it is received properly at TTI #3. If there is no correct transmission after three retries (delay column shows a value of 4) the data contained in the original TTI will be dropped. Also, note that the trace files are for the UL direction and as such no limitations are put in the DL direction. The trace information is stored in a binary file as unsigned 2 byte integers, in successive corresponding quadruples of stamp, payload, status, and delay values.

For the emulation, the transmission process is basically seen from IP layer, which can be assumed to be immediately above the RLC layer for our purposes here. If transmission of a

TTI fails because the number of retransmissions exceeds the maximum allowed, packet delivery depends on the RLC layer mode setting (Transparent (TM), Unacknowledged (UM), Acknowledged (AM)). In AM mode there are additional ARQ mechanisms to ensure that data can be delivered to upper layers. If these are absent (TM, UM), or fail (maximum number of retransmissions at the RLC layer has also been exceeded), the upper layer packet, which is partially contained in the concerned TTI is simply discarded (in practise, a timer based discard for upper layer SDUs is also used). In the emulator implementation, the RLC layer is emulated in UM mode without timer based discard and no (explicit) RLC/MAC protocol overhead.

The emulator intercepts the packets of the concerned application and stores them in its buffer. For each packet, an object is created that has two counters, *sent\_bits* which accounts for the amount of bits that are transmitted, and *delivered\_bits* which refers to the number of bits of that packet that are successfully received.

Each TTI, the emulator reads the allocations for that TTI from the trace, and if the status of the TTI is a success, it will account the specified amount of bits in the TTI as transmitted from the packet at the head of its buffer by incrementing the *sent\_bits* accordingly. Note also that if we have more than one packet in the buffer and if the amount of allocated bits in the TTI allows it, parts from several packets can be incorporated in one TTI (and as such, the *sent\_bits* of more than one packet can be modified at the same time).

During the next TTI, that is when the TTI get delivered, a similar update will be made on the *delivered\_bits* counter(s) of the concerned packet(s). When the *delivered\_bits* counter of a packet equals the packet size, the packet is fully delivered, and as such the emulator will release it to the destination.

On the other hand, if the status of a TTI is a failure and if the limit of maximum retransmissions is not reached, the emulator will not increment the *delivered\_bits* counter, as that signifies correct reception of the transmitted portion. If a further retransmission succeeds before we reach the maximum retransmission limit, the *delivered\_bits* counter is updated. However, if the maximum retransmission count is reached before the portion of the packet(s) contained in a transport block are correctly delivered, the packet(s) that is/are associated with the transport block are dropped from the emulator buffer.

### 3 EMULATOR GUI

Figure 2 shows the emulator GUI. The GUI is implemented as a Java™ applet and can be opened through any standard web browser or Java applet viewer. The GUI is divided into four main components. The first part, on the left, shows the cellular network layout in which the emulation is being undertaken and the location of the user within the concerned (green) cell. The user can be moved within the green cell either by dragging and dropping it to the new location or by simply clicking somewhere within the green area. Whenever the user's location is updated, the GUI sends the new location to the emulator, whether the emulation process is ongoing or not, so that the emulator can load and start using the appropriate trace file.

At the bottom of the cell layout, there are options that control the cellular environment as well as other emulation parameters. The environment option buttons control whether the emulation is for a micro or pico cellular environment. Selecting the pico cellular environment changes the environment to the one shown in Figure 3. In this figure, still the user can be dragged anywhere within the green area (the new pico cell).

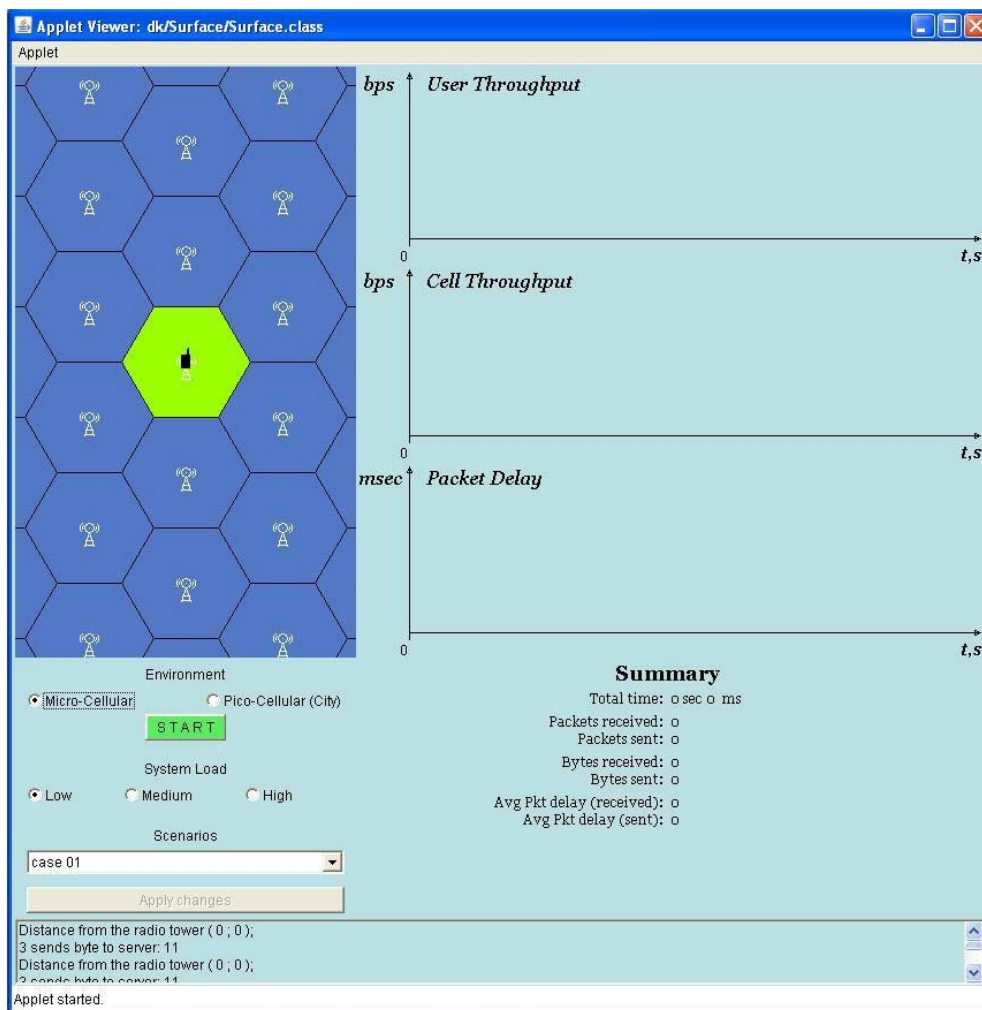
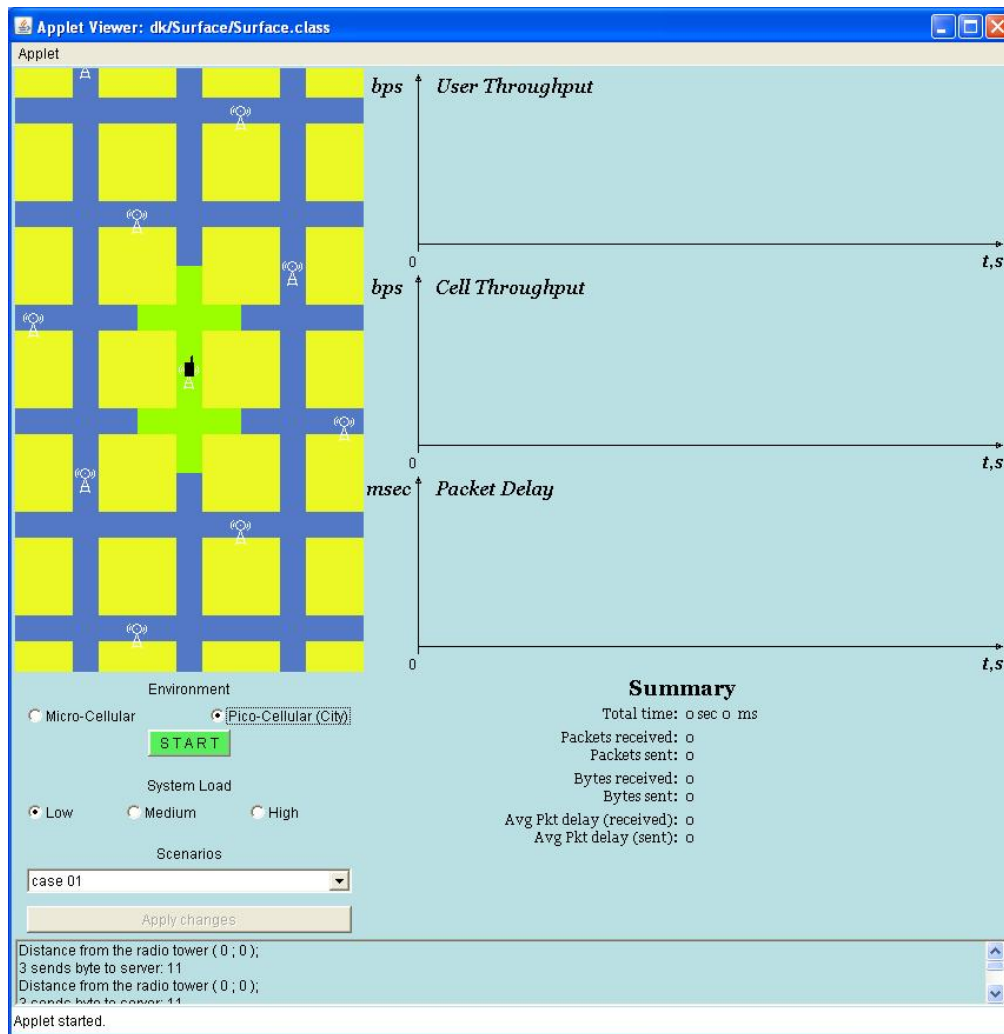


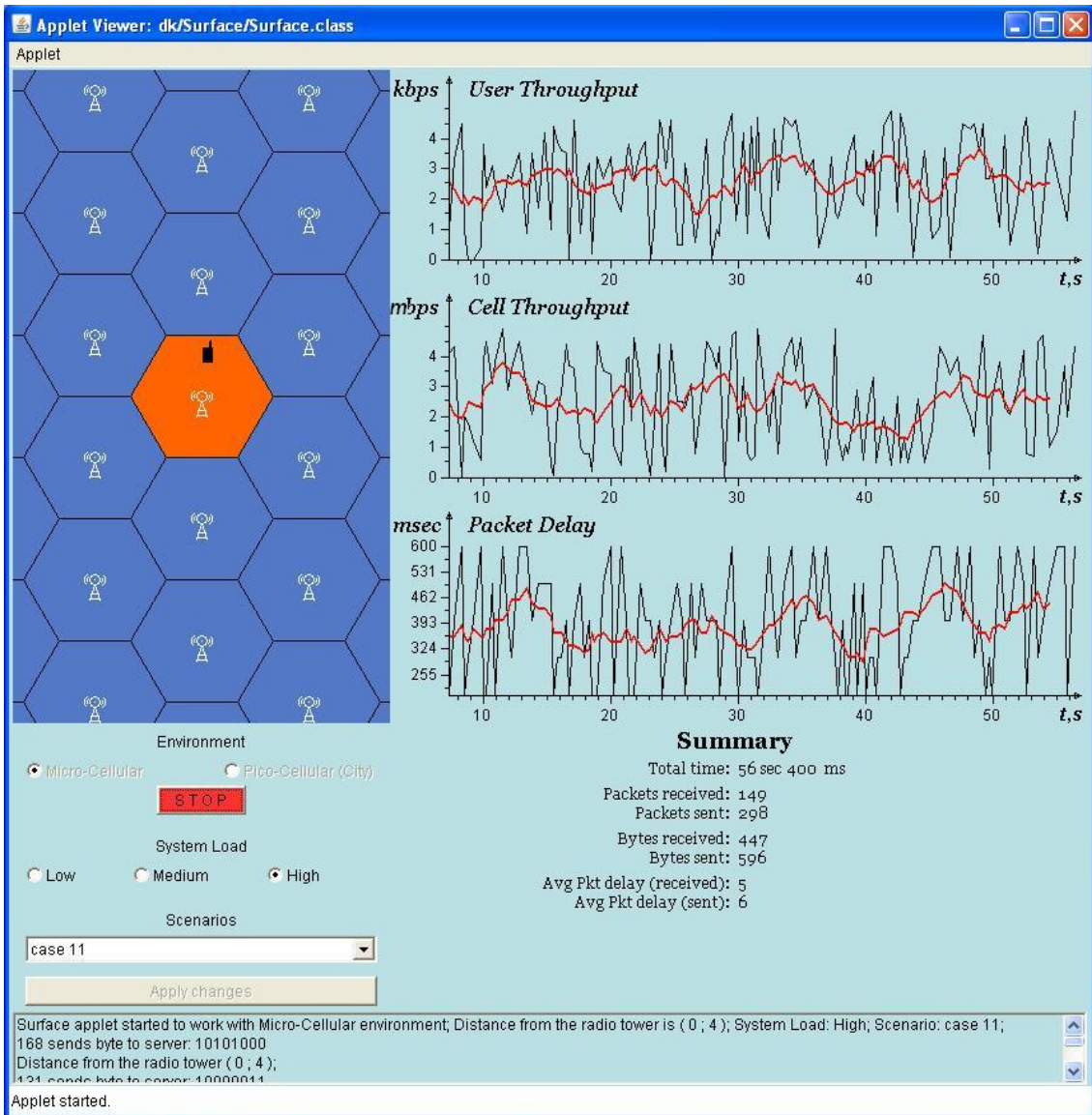
Figure 2: Emulator GUI, under Micro-Cellular setting.



**Figure 3: Emulator GUI, under Pico-Cellular setting.**

Apart from the cellular environment, the GUI can be used to change the system load or different use case scenarios (such as MIMO configuration) under which different system-level simulations are undertaken and hence trace files are available. The load and the scenario can be changed while emulation is ongoing. This is done by pressing the greyed out “apply changes” button, which becomes active whenever the load or the scenario are changed after an emulation has been started. However an ongoing emulation has to be stopped and restarted if we want to change the cellular environment.

Once the proper settings have been chosen, the emulation process can be started by pressing the START button. After this, any application such as a web browser or a streaming client can be started and all the packets that are exchanged between the client machine where the emulator GUI is running and the server accessed via the emulator are intercepted by the emulator and appropriate action is taken as described briefly in the previous chapter. The interface then will look like the one shown in Figure 4. As shown in the figure, the START button is now changed into a STOP button and the environment options are disabled. The cell is now shown in orange instead of green as a high load setting was chosen (the cell will be yellow if the load is medium).



**Figure 4: GUI once the emulator has started running.**

The right part of the GUI shows real time evolution of the key performance indicators of throughput (user and cell throughput), and packet delay. Both the instantaneous (black curves) as well as moving average values (red curve) are provided. A summary is also given below the curve about the emulation duration and the number of packets and bytes received and sent, as well as the average sent and received packet delays.

As such, the emulator GUI is not required. The emulation process can be started from command line where the key performance indicators can be logged into files for further processing. However, the emulator GUI provides a way to easily get the feeling of the overall emulation procedure and get immediate feedback to test the performance of the scenario under investigation while at the same time get a subjective feeling of the application that is generating the packets. Also the GUI makes it easy for remotely controlling the emulation process if the performance investigator does not have the right privileges to directly access the emulator machine.

## 4 PERFORMANCE EVALUATION

As mentioned previously, both objective and subjective performance evaluation of application quality when provided through the SURFACE air interface can be undertaken using the emulation platform. In this report, we present the objective evaluation that is carried out using FTP file transfers and both objective, and to some extent, subjective evaluation for streaming services. First the investigated scenarios will be described, followed by the performance results.

### 4.1 Investigated Scenarios

The trace files used for the evaluation were generated for the SURFACE uplink scheme 1 [7]: Scheme 1 uses Unitary Trace Orthogonal Design (UTOD) for the uplink transmission and a cell-based radio resource management (RRM) based on the Adaptive Transmission Bandwidth (ATB) concept in SC-FDMA air-interface [8][9]. The traces were generated for single user (SU) transmission with a  $2 \times 4$  MIMO configuration in SURFACE micro-cell scenario at a load corresponding to 12 users per cell.

Three trace files were generated, based on tracking three different quasi-stationary users: a user located near the base station (below 50 m with path loss in the range 80-100 dB), another one located at the cell center (approx. 100 m distance from the Base Station (BS) with path loss of approx. 115-120 dB), and a user close to the cell edge (above 170 m with path loss of approx. 125-130 dB). Henceforth, the three users are referred to as “near user”, “center user” and “edge user”. The traces contain the information for 10 seconds of simulations (20,000 TTIs), and during the emulations, the traces are simply wrapped around to the beginning when we reach the end of a trace.

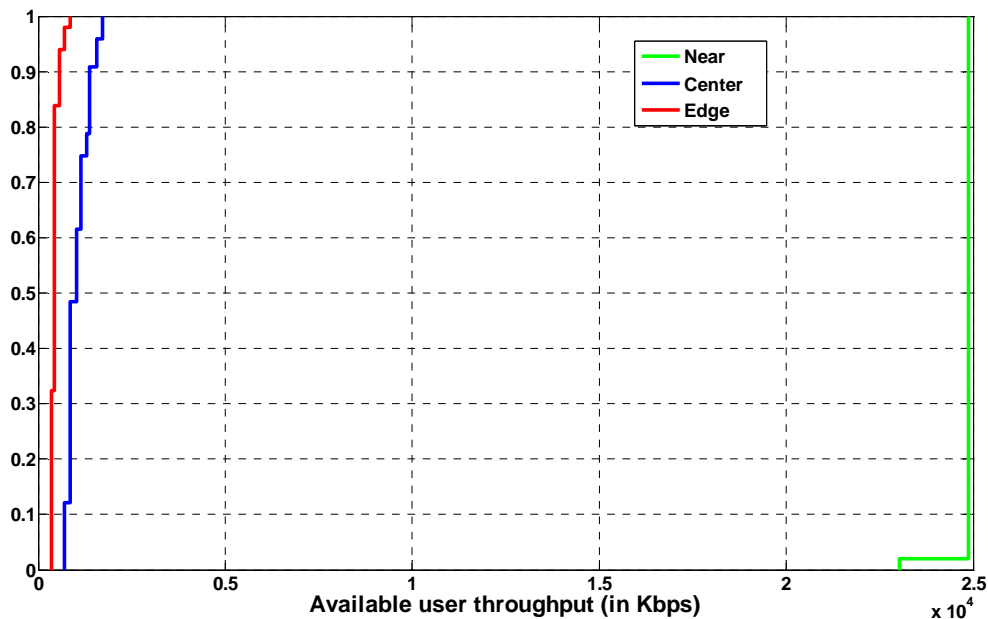
Table 2 shows the retransmission ratios for the three trace files used. As can be seen from the table, the near user experiences no drops of transport blocks, and only in less than 1.7% of the cases, a transport block is not received during its first transmission. On the other hand, for the edge user, more than 12% of the transport blocks have to be retransmitted, and 0.2% of them have to be dropped.

Figure 5 shows the cdf of the average allocated user throughput for the three trace files. Note that the throughput is the gross throughput that is available to the user and both failed and successful TTIs are taken into account. As can be seen from the figure, the near user experiences almost 24 times more throughput than the center user, while the center user has 2.5 times more throughput than that of the edge user.

**Table 2: Percentage retransmission of transport blocks as a function of user location**

	Near	Center	Edge
<b>No retransmission</b>	98.31	90.48	87.71
<b>1 retransmission</b>	1.68	8.27	10.33
<b>2 retransmissions</b>	0.01	1.00	1.38
<b>3 retransmissions</b>	0	0.20	0.38

Drop	0	0.05	0.2
------	---	------	-----



**Figure 5: cdf of available user throughput for different user locations**

Two different applications have been chosen for the investigations: FTP file download and video streaming. The FTP results are captured by using the command line Windows™ ftp command. The TCP on the Linux machine is configured for optimal performance, according to the recommendations in [4], specifically, TCP SACK and Eifel algorithm (TCP timestamps) are enabled.

For the streaming service, the Darwin Streaming Server™ [5] is used as a server and VLC (VideoLan Client) [6] is used as the client application. No modification has been made to the default configuration of these two applications, however, in order to avoid buffer overflow during the transfer of high bit rate streaming data, the receive buffer socket size of iptables (described in [2]), has been increased to the maximum allowed on the server (as can be determined using the Linux command `/sbin/sysctl net.core.wmem_max`) by using the `setsockopt` call.

## 4.2 FTP

Three different file sizes of 100KB, 1MB and 5MB were used for the FTP investigation. Table 3 shows the throughput realized by users at the three different locations when uploading the files of the specified sizes. As expected, the near cell user performs much better than center and cell edge users. Also, the ratio between the realized throughputs for the three different locations follows the same pattern as in the case of available throughput depicted in Figure 5.

In the case of near and center users, the throughput increases as the file size increases. This is because as the file size increases, the TCP connection will spend most of its time in full buffer state [3]. That is, it will be able to reach its full delay bandwidth product, and as such will be in continuous transmission mode, while with small file sizes, the time spent in slow start is considerable as compared with the total upload time.

**Table 3: Average FTP file download throughput values (in kbps) for different file sizes for users at different locations.**

	Near	Center	Edge
<b>100KB</b>	17430	699	321
<b>1MB</b>	18857	836	306
<b>5MB</b>	19883	836	303

On the other hand, for the case of the cell edge user, there is a slight decrease in the throughput as the file size increases. This can be explained by looking at the packet drop rates, as shown in Table 4. As can be seen in the table, there is no packet drop in the near case, and the drop rate for the center user is negligible, while the drop rate for the cell edge user is considerable. So this increase in the drop rate counteracts on the performance improvement due to the full buffer utilization.

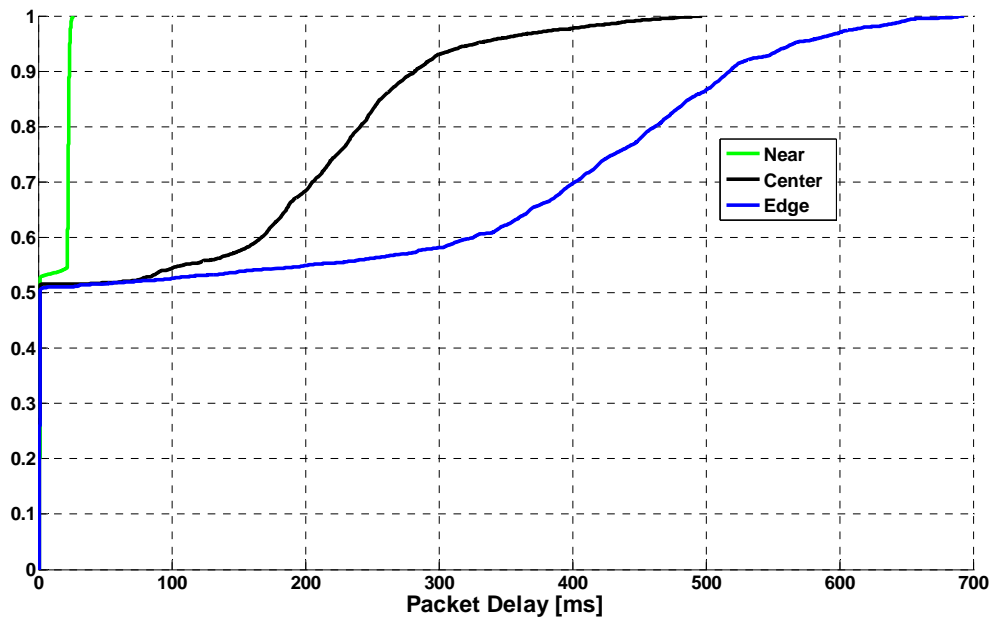
**Table 4: Experienced IP packet drop rates (%) for different file sizes for users at different locations.**

	Near	Center	Edge
<b>100KB</b>	0	0	0
<b>5MB</b>	0	0.12	1.6
<b>5MB</b>	0	0.12	1.97

Finally, Figure 6 shows the packet delay experienced for the three user locations while downloading a 5MB file. The 5MB file case is chosen in order to collect enough statistics for the cdf plot. The maximum delay experienced by an IP packet for the near user is around 27 ms while for the center and edge users, the delay reaches up to 500 and 700 ms, respectively. This shows that for real time interactive services, the center and edge users can suffer greatly from performance degradation as the packet delays are well over the acceptable packet delay by an interactive real time service [4].

### 4.3 Streaming

For the streaming performance investigations, three different video streams are used. The first two are the sample\_100kbit.mp4 [size: 926KB, length: 70sec, low resolution, encoded bit rate of 100kbps] and sample\_300kbit.mp4 [size: 3.2MB, high resolution, length: 70sec, encoded bit rate of 300kbps] sample files that are supplied by Apple™ for testing Quicktime™ player and Darwin Streaming Server. A very large streaming video file, which is the trailer for the movie “Shark Tale” [size: 62MB, length: 139s, very high quality and resolution], is also used. This file is variable bit rate (VBR) encoded of up to 12Mbps instantaneous rate, but with an average around 3.7Mbps as can be detected by the VLC media player.

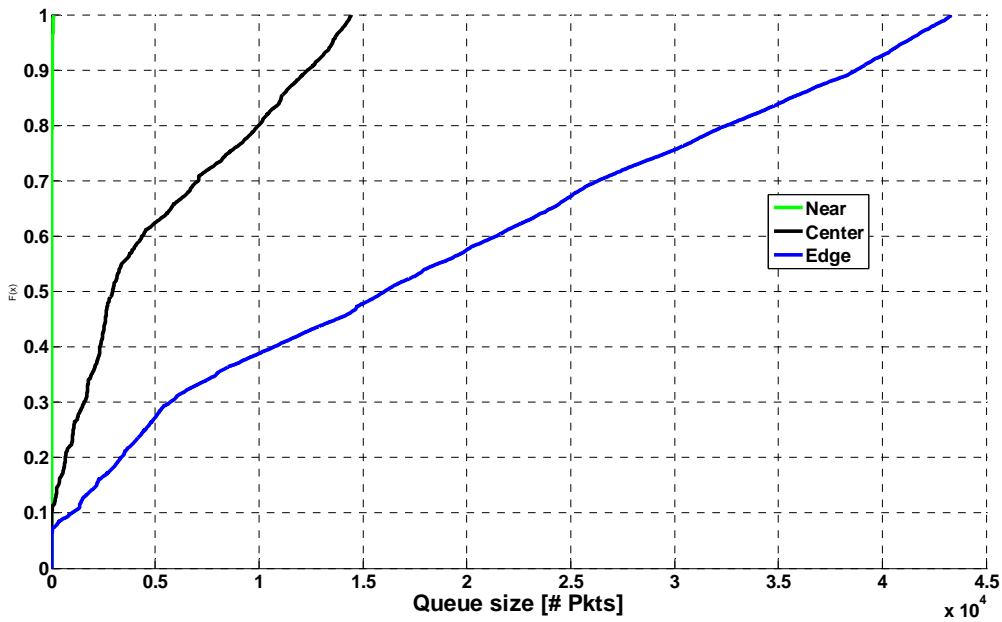


**Figure 6: cdf of packet delay, for 5MB file download.**

For the near user scenario, all the three streaming videos were played flawlessly without any interruption or buffering, and the results were indistinguishable from that of streaming the videos without any emulator in between. For the center user, the sample\_100k and sample\_200k streams were also played flawlessly. For the edge user, even though there were also no buffering interruptions for these two streams, watching the video was not as pleasant as the other two users, as there were few very short freezes, lasting few hundred ms, due to lost frames. The playback of the Shark Tale stream did not finish for both the center and edge users, as the encoding rate was too high and buffer overflow occurred during the first few seconds of playback.

Figure 7 shows the queue buildup during the streaming of the Shark Tale trailer video for the three different user locations. While the maximum queue size for the near user is less than a 168 packets, the cell edge user experiences queue size of up to 43,000 packets. This high value is mainly due the nature of UDP protocol that is used for the real time streaming, and the server will not realize data is not received at the client and will simply keep pushing data to the client till it has finished sending all the streaming data. This excessive buildup of the packet queue in the emulator leads to further drops by iptables and hence the freezing of the video stream.

Table 5 shows the average throughput values experienced during the streaming of the different videos for the different user locations. It can be noted that for the sample\_100k and sample\_300k streams, the performance in terms of throughput is very similar for the three users, owing to the constant bit rate flow which is well within the available throughput for each case (see Figure 5). However, due to the limitation in the available throughput, the center and edge users were not able to realize the required bit rate that is needed for a proper playback of the video stream.



**Figure 7: cdf of packet queue size for the Shark Tale video stream.**

It is not only the bit rate that is bringing down the performance for the center and cell edge users, but also the IP packet drop rate, which is depicted in Table 6. This drop rate is only referring to the packets that are dropped due to the decision of the emulator according to the trace file, and not due to iptables queue build up illustrated in Figure 7.

**Table 5: Experienced average throughput values (in kbps) for the different streaming videos.**

	Near	Center	Edge
sample_100k	92	91	90
sample_300k	330	329	304
Shark Tale	3528	360	306

**Table 6: Experienced IP packet drop rates (%) for the different video streams**

	Near	Center	Edge
sample_100k	0	0.18	1.5
sample_300k	0	0.22	3.7
Shark Tale	0	0.3	4.3

Figures 8 and 9 show the cdf of the packet delay for the 100kbps and 300kbps files, respectively. It can be seen that the delays are even more for the 300kbit streaming as compared with the delays of Figure 6 when downloading the 5 MB file. This is because, even though the total bytes transferred is less in this case (file size is 3.2MB), as the transfer is done using UDP there is no flow control as in TCP, and the queue builds up. The maximum queues for the near, center and edge users for the 300kbps streaming case are 16, 22 and 216 respectively. Fortunately, the queue sizes are still within the limits of what iptables and the emulator can handle, and as such the video streaming is experienced with good quality. However, if the concerned service was an interactive streaming service such as video conferencing, the user experience would have been much worse because, even though data is not dropped, the two communicating entities will be out of sync due to the queue build up and the resulting high packet delay.

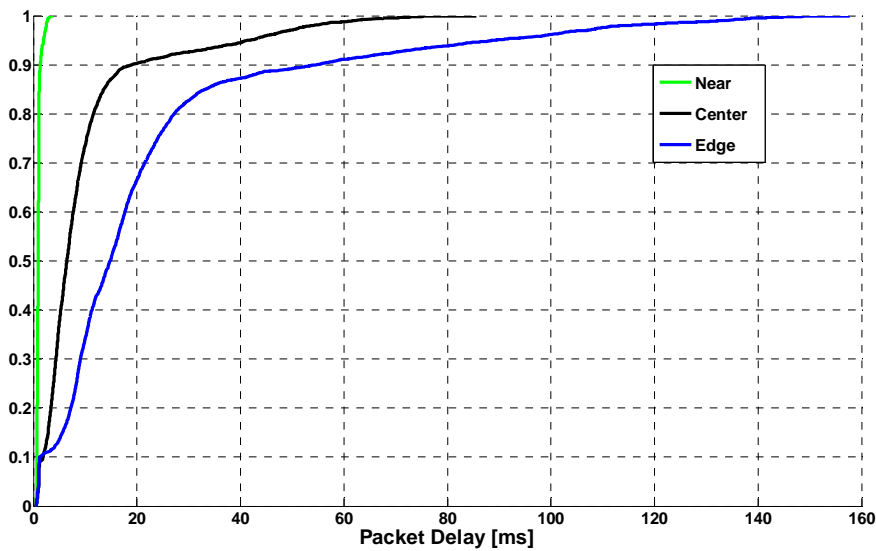


Figure 8: cdf of packet delay, for sample\_100kbit streaming

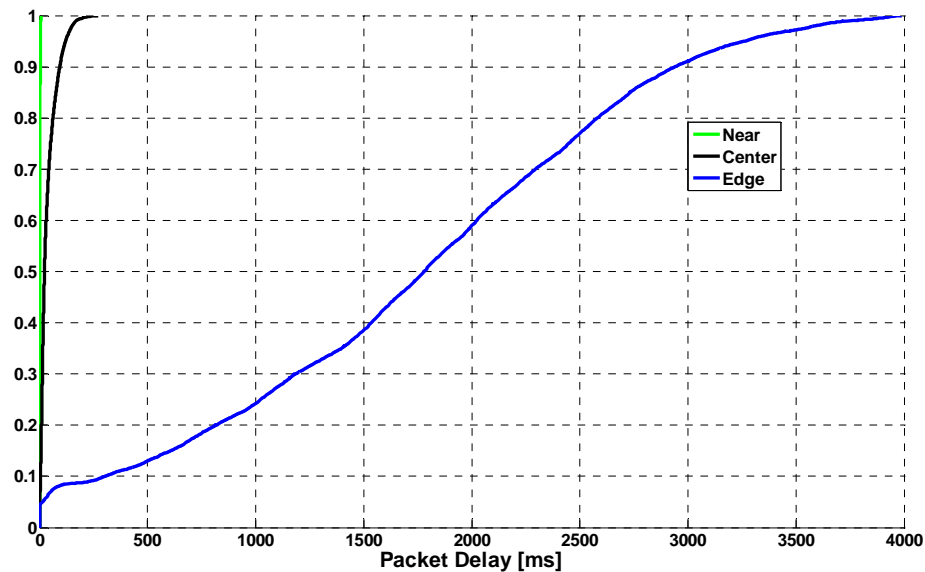


Figure 9: cdf of packet delay, for sample\_300kbit streaming

## 5 CONCLUSION

In this deliverable, description of the SURFACE emulator - system-level simulator coupling and the GUI that is used to control and track the emulation process is described. Preliminary performance investigations of FTP file transfer and streaming services are also given. Users located at different distances from the base station were investigated, for file transfers of different sizes as well as for real time streaming of videos of differing quality. From the preliminary results it is concluded that the SURFACE unified air interface gives good performance for FTP file transfers and streaming of reasonable size videos, regardless of the user location. However, only users close to the base station were able to experience good quality video streaming when the concerned streaming file was quite high.

## 6 REFERENCES

- [1] IST-2006-27187 SURFACE, “Annex I – Description of Work”, Jan. 2006.
- [2] IST-4-027187-STP-SURFACE, D7.2 v1.0, “Design specification of the air interface emulator”, Apr. 2007
- [3] Richard Stevens, “TCP/IP Illustrated, Volume 1: The protocols”, 1994
- [4] Oumer Teyeb, “Quality of Packet Services in UMTS and Heterogeneous Networks: Objective and Subjective Evaluation”, PhD Thesis, Dec 2006
- [5] Darwin Streaming Server, <http://developer.apple.com/opensource/server/streaming/>
- [6] Video LAN Media Player, <http://videolan.org/vlc>
- [7] IST-2006-27187 SURFACE, “D7.3 – First results from link and system level simulations (internal)”, v2.0, February 2008.
- [8] IST-2006-27187 SURFACE, “D5 – Description of the network Optimal Transmit and Receive“, v1.0, January 2008.
- [9] IST-2006-27187 SURFACE, “D6 – Transmit and Receive Architecture Development and Algorithmic Implementation“, v1.0, July 2008.