Multimedia Tools and Applications (2021) 80:19107–19135
https://doi.org/10.1007/s11042-021-10708-3

Protocol-agnostic method for monitoring interactivity time in remote desktop services

Jesus Arellano-Uson [1] · Eduardo Magaña [1,2] · Daniel Morató [1,2] · Mikel Izal [1,2]

Received: 13 April 2020 / Revised: 21 October 2020 / Accepted: 10 February 2021 /
Published online: 24 February 2021

Abstract
The growing trend of desktop virtualisation has facilitated the reduction of management costs associated with traditional systems and access to services from devices with different capabilities. However, desktop virtualisation requires controlling the interactivity provided by an infrastructure and the quality of experience perceived by users. This paper proposes a methodology for the quantification of interactivity based on the measurement of the time elapsed between user interactions and the associated responses. Measurement error is controlled using a novel mechanism for the detection of screen changes, which can lead to erroneous measurements. Finally, a campus virtual desktop infrastructure and the Amazon WorkSpaces solution are analysed using this proposed methodology. The results demonstrate the importance of the location of virtualisation infrastructure and the types of protocols used by remote desktop services.

Keywords Remote desktop · Interactivity · Response times · Services in the cloud

✉ Jesus Arellano-Uson
jesus.arellano@unavarra.es

Eduardo Magaña
eduardo.magana@unavarra.es

Daniel Morató
daniel.morato@unavarra.es

Mikel Izal
mikel.izal@unavarra.es

[1] Department of Electrical, Electronic and Communications Engineering, Public University of Navarre, Pamplona, Spain

[2] Institute of Smart Cities, calle Tajonar 22, 31006 Pamplona, Spain

1 Introduction

Traditional system infrastructures with equipment running applications locally on desktop computers are evolving in favour of remote desktop systems. In this paradigm, a desktop computer is a computer characterised by reduced computational power and resources. Such a computer is used as a visualisation and peripheral control machine. This type of desktop computer is called a 'thin client' and it behaves like a visualisation machine, rather than a computing machine. Operating systems and applications run on a remote server. This server can be located inside or outside a user's network and is responsible for processing inputs (keyboard strokes and mouse clicks) from thin clients and sending representations (screen updates) through a network to thin clients (display hosts).

Remote desktop solutions provide flexibility and facilitate the deployment of custom instances for user profiles that may require specific environments. Additionally, they simplify management tasks by facilitating the mobility of users and security provisioning. Users have the ability to access remote desktops from different devices, such as personal computers, tablets, or mobile phones. This type of infrastructure facilitates the reduction of the underlying costs of a computer seat, such as costs related to management and maintenance, as well as those associated with the deployment and configuration of equipment.

There are several different ways to implement a remote desktop scheme, and they can be implemented for different purposes. Some schemes are specialised to perform remote assistance functions, in which the use of remote desktop technology is implemented in a timely manner for assistance tasks (where a device is managed remotely) or desktop streaming (where a device is managed locally, but displayed remotely). Some solutions belonging to this category are TeamViewer [34], Google Remote Desktop [9], and LapLink [19]. In contrast, there are many other solutions designed for continuous use, such as Windows Remote Desktop [37], Real VNC [29], and Apache Guacamole [3]. There are different protocols that enable these services, such as PCoIP [35], the Remote Frame Buffer protocol [30], and Remote Desktop Protocol (RDP) [26]. Some of the main characteristics of these protocols are discussed in [20]. In this type of solution, computer resources are typically centralised, and remote desktops are virtualised. In this manner, a single host can serve multiple desktops simultaneously in what is known as a Virtual Desktop Infrastructure (VDI) [33].

In this kind of infrastructure, different desktop virtualisation solutions are made available through a public cloud. Amazon provides this type of infrastructure through the cloud-managed Amazon Web Services system [1], where the globally accessible service Amazon WorkSpaces [2] is offered. This solution allows clients to select the required characteristics for their virtualised instances, such as memory, storage, CPU, and operating system. These types of platforms are provided as a suitable alternative for both personal and business clients that seek to outsource the management of desktop virtualisation infrastructures. The outsourcing of computing resources is becoming increasingly common. It is intended that a client can access services using inexpensive mobile equipment and pay for the use of more powerful computing resources. An example of the growth of this type of strategy in other sectors is Google Stadia [10], which is a video gaming platform that allows users to access different video games through a streaming service. Such VDI solutions are relatively common in medium and large enterprises.

The ultimate goal of a remote-desktop-based system is for users to interact with a solution as if it is a local computer. This requires that the experience perceived by users be conditioned as little as possible by the fact that the machine that is executing applications is not physically

located in the vicinity of clients. This requires a high-speed and low-latency network in addition to the highest possible availability of infrastructure. The current migration trend towards this type of remote desktop virtualisation infrastructure makes the development of new analysis and measurement tools essential. This paper proposes a novel measurement technique for the characterisation of VDI performance in terms of interactivity time for remote desktop systems. Interactivity time is defined as the time elapsed between a user performing an interaction (keystroke or mouse click) and the corresponding effect being displayed on the screen. Our proposed methodology improves actual solutions that measure interactivity time in the following ways:

- The ability of real-time measurement with real users and applications. A controlled scenario or instrumentalization of user applications is not required.
- Measurement of real interactivity experienced by users as the time elapsed between a real physical interaction and the corresponding final screen update.
- Measurement of user interactivity not on the server, but also on the client to reflect real user experiences.
- A stability check procedure that is able to distinguish time periods when measurement is not reliable.
- Recording of screen updates to identify the points in an application workflow where problems are present.
- Evaluating the performance of real infrastructures in local and public VDI solutions.

Additionally, this methodology can be applied to the characterisation of user satisfaction with online interactive services, such as the Office 365 suite, Google Docs, or any other interactive service accessible through a web browser or specific application. It can also be applied to new gaming services, such as Google Stadia, with convenient hardware deployment.

The remainder of this paper is organised as follows. Section 2 presents the elements of a remote desktop system and discusses metrics that can be extracted to evaluate the experiences perceived by users. Section 3 describes the proposed methodology and discusses how it can be applied to measure thin client interactivity. Section 4 presents experiments that were conducted to verify the feasibility of our methodology. Section 5 presents the analysis of different types of virtualisation infrastructure and a comparison among them. Section 6 discusses related work presented in the literature. Finally, conclusions and future work are discussed in Section 7.

2 Strategies for measuring interactivity in remote desktop systems

Remote desktop scenarios typically consist of a thin client device with reduced computational resources and a remote server machine that provides a virtualised desktop. Professional and domestic users are accustomed to utilising desktop computers in a local environment, in which the perceived delay between actions (with a keyboard/mouse) and the response on a screen depends solely on their local equipment. However, the general trend of virtualising desktops adds new sources of delay to traditional scenarios based on network performance metrics and the sharing of server resources among several virtualisation instances.

Traditional monitoring metrics analyse network parameters such as packet delay, jitter, bandwidth, and loss. However, these metrics characterise only the network segment, that is Quality of Service (QoS) and are not indicative of the quality perceived by the user, that is,

Quality of Experience (QoE), because they do not consider all of a system's elements. The extraction of metrics that reflect the influence of the current states of all the elements is complex if the process of obtaining such metrics requires the analysis of the protocols used for communication. Protocol analysis would facilitate measuring the delay between requests and responses for VDI messages at the network level. However, some VDI implementations, such as group client requests, emulate interaction responses before they are actually processed by the server or use ciphers for network traffic. Therefore, this approach is difficult to generalise and in some cases, based on protocol implementations, it is unrealisable [23].

The goal of this study was to obtain a QoS metric that provides information regarding the usability of a service based on the actions performed by clients, which are closely related to the QoE perceived by users. The overall behaviour of a system depends on all of the elements that comprise that system. The QoE perceived by users is constrained by the delay between keyboard or mouse interactions and the on-screen representation of the corresponding responses to those interactions. For example, a response could be the opening of a new window, a change in the colour of a button, or a new character appearing in an input box. This type of delay is referred to as interactivity time and is represented in Fig. 1. The proposed methodology aims to extract this metric and requires software to be deployed on a user device (thin client). Therefore, it is necessary to evaluate user interactions with their local devices (t_e) and to monitor user graphic environments to determine when the corresponding responses are represented on local devices (t_C).

The methodology proposed in this paper is intended to be able to measure and characterise the interactivity of users in remote desktop environments, as well as in online interactive services, such as office applications accessible through a web browser. A tool that monitors the usability of a VDI structure should provide the necessary context metrics for interactivity measures obtained in such a manner that they allow for the determination of whether deterioration in interactivity time is caused by a shortage of client computational resources, network behaviour, or VDI server behaviour. Therefore, metrics such as CPU, RAM, network usage, or even screenshots can be extracted to facilitate contextual analysis.

3 Methodology for calculating interactivity time

To perform interactivity time measurements, we evaluate the design of a tool running on a thin client that frequently checks a user's screen and saves its state. When a user clicks the mouse or presses a key (timestamp t_e in Fig. 1), the recorded state represents the screen content prior

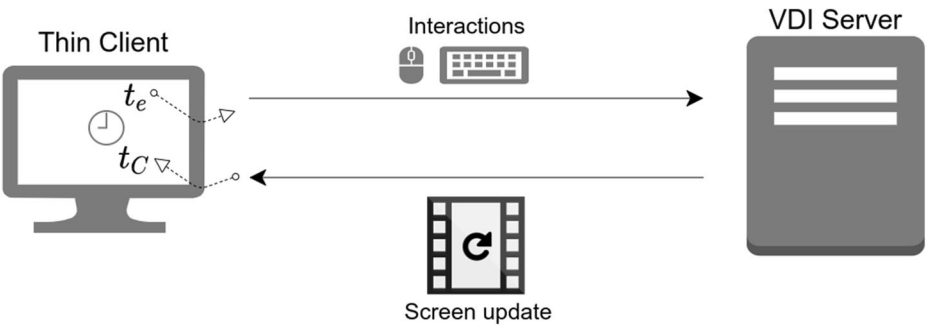

Fig. 1 Scheme depicting how interactivity time is obtained

to the event. The VDI client then sends the user action to the server and successive checks of the client's screen are performed until an on-screen change is detected (timestamp t_C in Fig. 1), which completes the user interaction.

The difference between the timestamps t_C and t_e is the interactivity time. It can be mapped to a mean opinion score (MOS) scale using the ITU-T G.1030 [14] recommendation, which is designed to estimate end-to-end performance in IP networks for data applications. Additional instrumentation on the client can be used to monitor its status and extract additional contextual measures that facilitate the identification of the root causes of service degradation, such as CPU or memory utilisation.

The thin client through which a user interacts is responsible for representing the graphical responses sent by a server. Therefore, this methodology implies that the proposed software must be executed on the thin client because it is the end point that combines all of the accumulated delays of an infrastructure. An implementation of the proposed methodology called Thin Client Latency Analysis (TeCLA) was conducted to evaluate its feasibility and accuracy. AutoHotkey (AHK) [4] was selected as the development framework. AHK is a free open-source framework designed for the Microsoft Windows operating systems. It facilitates the creation and automation of user tasks, as well as interaction with the Windows API. AHK code is semi-compiled to reduce memory consumption and improve runtime performance. TeCLA runs on a thin client to measure the interactivity times for all actions performed by a remote desktop user in real time.

In the following subsections, we describe the proposed measurement algorithm and analyse its main elements, namely a screen change detection procedure and instability detection mechanism to detect erroneous measurements.

3.1 Detection of screen changes

Different schemes for detecting changes in screen bitmaps were tested during the development of our method, including classical signal processing techniques. The problem with such strategies lies in the computational time required. Processing bitmaps can lead to the queuing of measurement requests, which negatively impacts real-time operation and reduces the accuracy of recorded timestamps. This could interfere with a user's experience by influencing thin client CPU utilisation and increasing the extracted interactivity time. Screen analysis could be delayed by storing bitmaps on a disk with sufficient frequency to obtain sufficient resolution for interactivity measurements. However, the time required to write bitmaps to the disk introduces additional resource consumption that distorts the measurement of interactivity times.

The adopted approach performs real-time comparison of screen bitmaps by employing cyclic redundancy checks (CRCs) [28], specifically CRC-32 [12]. Cyclic codes are frequently used in computer science for error detection. Their computation can be performed rapidly, and they are commonly implemented using operating system calls at a system's kernel level. The calculation of a CRC is performed on an array of pixels that constitute a raster graphic on the screen, and the comparison of two screen bitmaps is performed by comparing their respective CRCs. The CRC from the bitmap recorded immediately before a user event is compared to all following screen updates. If a difference is detected, it is considered that the response from the server has been delivered and displayed. Compared to the aforementioned alternatives, the use of CRCs is more efficient in terms of time complexity. Therefore, this method facilitates

continuous screen checks and real-time analysis. The greater the number of checks performed per unit time, the higher the resolution that can be obtained for measurement.

The proposed methodology requires continuously extracting bitmaps from a graphical environment and processing them. However, continuously capturing full-screen bitmaps may require excessive effort for high screen resolutions. Figure 2 presents the cumulative distribution function (CDF) of the time required to extract a bitmap and calculate its checksum based on the bitmap size. This study was conducted using a computer with two CPU cores running at 2.0 GHz with 8 GB of RAM, a Windows 10 Home installation (10.0.17134), and a screen monitor with a refresh rate of 60 Hz. This means that every $T_f = 1/60Hz = 16.6\ ms$, there is a screen update. It is not useful to make more than one call to extract a screen bitmap per screen refresh period T_f. Therefore, the operating system postpones the response to the requested call until the next refresh period (up to 16.6 ms later for a 60 Hz refresh rate), meaning the cost of the process tends to be accumulated around values close to 16.6 ms. The processing time is influenced by when the call is issued within the refresh period, the system call response time, and the time required to calculate the checksum of the extracted bitmap. The CDF is approximately linear, meaning its value is largely determined by when the screen capture is requested relative to when the next screen update is performed.

Figure 2 illustrates how the processing time increases for larger bitmap sizes. There is a compromise between the size of the evaluation area and the impact on CPU utilisation on the thin client. The choice of the screen area used for comparisons must be considered to expedite this process. In cases with limited client resources, only a restricted region of the screen can be captured and analysed. This evaluation area will be centred on the current user mouse position. This approach can efficiently detect changes on the screen that take place around the same area when the associated actions are performed.

The complexity of the capture and the analysis of a screen bitmap do not depend only on the size of the bitmap, but also on its colour depth, meaning the number of bits required to define the colour of a pixel. Fig. 3 presents the CDF of the time required for the extraction and computation of the checksum for a bitmap consisting of 683 × 364 pixels. The equipment on which these measurements were performed comprised a graphics adapter with a true tone configuration for colour depth using 32 bits per pixel. The time required to generate bitmaps increases significantly as the number of bits required to define pixel colouration increases. Therefore, any implementation of the proposed methodology should opt for the lowest colour depth configuration available based on the graphics adapter in use (in this case, an Intel HD Graphics 4400-series graphics adapter with a base frequency of 200 MHz).

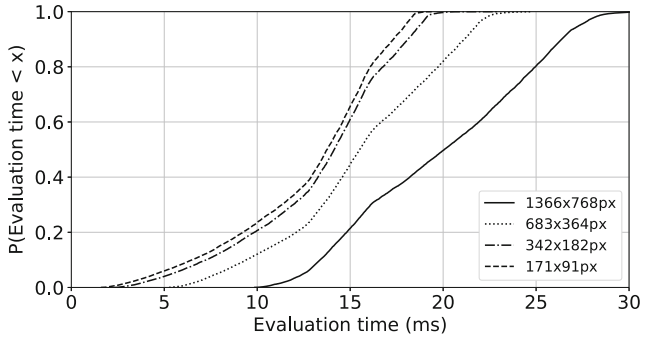

Fig. 2 CDF of the time required to capture a bitmap and compute its checksum

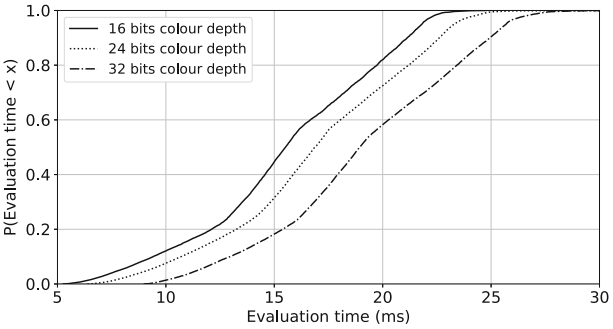

Fig. 3 CDF of the time required to extract and compute the checksum of a 683 × 364 pixel bitmap according to the colour depth configuration

Our analysis revealed a compromise between the dimensions of the evaluation area, colour depth, and time required to extract a bitmap and process its checksum. These parameters should be selected based on the CPU in use, as discussed in Section 4.

3.2 Screen instabilities

To measure interactivity time, we assume that screen changes are the consequences of user events (keyboard or mouse). However, sometimes, there may be autonomous changes in elements on the desktop (e.g., a clock blinking every second), in the current window (e.g., a banner containing an animation), or in a different window (e.g., video playing in the background). We use instability to refer to any screen change caused by such autonomous changing elements. Instabilities can lead to false positives in the identification of screen changes related to user events. If one of these changes occurs in the evaluation area shortly after a user event, it will be considered as the response to that event and an interactivity time shorter than the real time will be measured. We must identify such instabilities to flag the following measurements of interactivity times as unreliable.

Figure 4 presents a timeline in which the time instants when two instabilities occur are marked as $t_{I,\,1}$ and $t_{I,\,2}$. A user event occurs between these instabilities at time t_e. The interactivity time should be measured from t_e to the point when the feedback from this event is displayed on the screen at t_C. If $t_{I,\,2} < t_C$, then an instability occurs before the user event feedback, and the measured interactivity time is computed as $t_{I,\,2} - t_e$ instead of $t_C - t_e$. We evaluate the probability of such errors to facilitate the design of a control procedure that marks invalid measurements in the following section.

We model the occurrence of instabilities based on a stochastic description of instability time. We describe such instabilities using a random arrival process in which the time between any two adjacent instability events is described using independent and identically distributed (i.i.d.) random variables. These variables are contained in the set $\{Z_k\}$, where k represents the interval between $t_{I,k}$ and $t_{I,k+1}$. We consider the following two possible models for the random variable Z (representative of all i.i.d. Z_k values):

- The time between instabilities is constant with a period T_s. This scenario corresponds to events such as the periodic blinking of a clock on the screen.
- The instabilities are randomly spaced based on an exponential inter-arrival random variable with an average T_s. The parameter in the exponential random variable is $\lambda = 1/T_s$.

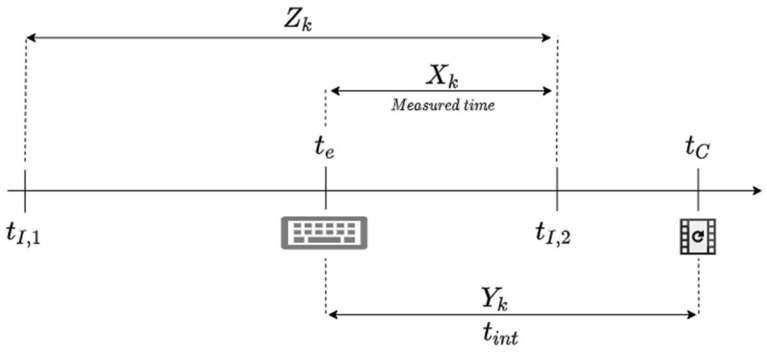

Fig. 4 Diagram of user events overlapping with instabilities

In Fig. 4, the random variable X_k represents the time between a user event and the next instability. If this value is smaller than the real interactivity time, then the measurement will underestimate the real value. We assume that user events are statistically independent instability events and that they occur in random epochs between two instabilities. These random epochs are uniformly distributed and cover the time interval between two instabilities. The variable X_k represents the residual life of the random variable Z_k. If $F_Z(x)$ is the probability distribution function of the random variable Z, then the probability density function of the random variable X (representative of the set $\{X_k\}$ of i.i.d. random variables) is given by Eq. (1) [17], where $E[Z]$ is the expected value of the random variable Z.

$$f_X(x) = \frac{1-F_Z(x)}{E[Z]} \tag{1}$$

We consider two models for interactivity time. For the probability distribution function $F_Y(y)$ of the random variable Y (representative of $\{Y_k\}$ i.i.d. random variables), we have the following:

- The interactivity time is described using a uniform random variable between r and m. This corresponds to a scenario with a round-trip time (RTT) between the client and server (minimum interactivity time) with a value of r. The parameter m controls the maximum interactivity time.
- The interactivity time is described using a shifted exponential random variable with an offset r. This offset corresponds to the RTT, and the exponential random variable models an unbounded interactivity time. We use μ as a parameter in the exponential random variable component. The expected value of this exponential random variable with an offset is defined in Eq. (2).

$$E[Y] = r + 1/\mu \tag{2}$$

We compute the probability of a measurement error as $P(X < Y)$, which represents the probability of the residual life of the instability inter-arrival time being smaller than the interactivity time. Both random variables X and Y are independent; meaning the probability

density function for the combination of both variables is simply the product of their probability density functions (Eq. (3)).

$$f_{X,Y}(x,y) = f_X(x) f_Y(y) \tag{3}$$

We compute $P(X < Y)$ by integrating $f_{X,Y}(x, y)$ in the area where $X < Y$. We consider four possible combinations of these two models for instability inter-arrival time and interactivity time.

3.2.1 Periodic instabilities and uniform interactivity times

The probability density function for the combination of these events is defined in Eq. (4).

$$f_{X,Y}(x,y) = \begin{cases} \frac{1}{T_s(m-r)} & r < y < m, x < T_s \\ 0 & otherwise \end{cases} \tag{4}$$

If the inter-instability time is greater than the maximum interactivity time ($m < T_s$), then the probability of a measurement error is defined by Eq. (5). This is the integration of the probability density function over the support of the uniform random variable for the interactivity time (from r to m) and over the range of values of the residual life of the instability period that comply with the restriction of being smaller than the interactivity time.

$$P(X < Y) = \int_r^m \int_0^y \frac{1}{T_s(m-r)} dxdy = \frac{m+r}{2T_s} \tag{5}$$

In this simple scenario, the probability of an error is simply the quotient of the average interactivity time (m + r)/2 and average inter-instability time T_s.

3.2.2 Periodic instabilities and shifted exponential interactivity times

The probability density function for this combination of events is defined by Eq. (6), and the probability of a measurement error is defined by Eq. (7).

$$f_{X,Y}(x,y) = \begin{cases} \frac{1}{T_s} \mu e^{-\mu(y-r)} & r < y, x < T_s \\ 0 & otherwise \end{cases} \tag{6}$$

$$\begin{aligned} P(X < Y) &= \int_r^{T_s} \int_0^y \frac{1}{T_s} \mu e^{-\mu y} dxdy + \int_{T_s}^{\infty} \int_0^{T_s} \frac{1}{T_s} \mu e^{-\mu y} dxdy = \\ &= \frac{r}{T_s} + \frac{1}{\mu T_s} \left(1 - e^{-\mu(T_s - r)}\right) \end{aligned} \tag{7}$$

3.2.3 Exponential instabilities and uniform interactivity times

The probability density function for this combination of events is defined by Eq. (8). Assuming that $m < T_s$, the probability of a measurement error is defined by Eq. (9).

$$f_{X,Y}(x,y) = \begin{cases} \frac{\lambda}{m-r}\mu e^{-\lambda x} & r < y < m, 0 < x \\ 0 & otherwise \end{cases} \tag{8}$$

$$P(X < Y) = \int_r^m \int_0^y \frac{\lambda}{m-r} e^{-\lambda x} dxdy = 1 - \frac{T_s}{m-r}\left(e^{-\frac{r}{T_s}} - e^{-\frac{m}{T_s}}\right) \tag{9}$$

3.2.4 Exponential instabilities and shifted exponential interactivity times

For this final model, Eq. (10) defines the probability density function and Eq. (11) defines the probability of a measurement error.

$$f_{X,Y}(x,y) = \begin{cases} \lambda e^{-\lambda x}\mu e^{-\mu(y-r)} & r < y, 0 < x \\ 0 & otherwise \end{cases} \tag{10}$$

$$P(X < Y) = \int_r^\infty \int_0^y \lambda e^{-\lambda x}\mu e^{-\mu y} dxdy = 1 - \frac{\mu T_s}{1+\mu T_s} e^{-\frac{r}{T_s}} \tag{11}$$

In the following subsection, we describe a procedure that checks for instabilities on the desktop and evaluates the reliability of interactivity measurements based on the results provided by the models described above.

3.3 Stability check

We designed a stability check procedure that identifies instabilities in the evaluation area during the time when a user is not performing any action. We define a stability measurement interval as the time interval between interactivity measures (from the end of one interactivity measure to the start of the next interactivity measure), between instabilities, or between an event in an interactivity measure and an instability. The stability measurement intervals are related to the time intervals when a user reads the information presented on the screen or performs other tasks away from the computer. In periods of time when the user is not performing actions, the screen is captured at a refresh rate T_f and the changes on the screen are analysed. Therefore, a stability measurement interval is a time interval without any interactivity measurements or instabilities.

If the stability measurement interval is sufficiently long, it is an indication of a large average inter-instability time, and the following interactivity samples are flagged as reliable. We estimated the probability of an error in interactivity time measurement caused by the appearance of an instability using different models (Eqs. (5), (7), (9), and (11)). Such errors depend

on the average inter-instability time. We can determine the minimum stability measurement interval T_s if it is representative of the expected value and yields a probability of a measurement error below a specified threshold.

Figure 5 presents the probabilities of a measurement error as estimated using the four models discussed in Section 3.2. We consider the following two scenarios:

- Local WAN VDI: An expected value of interactivity time of 100 ms with a minimum value of $r = 50$ ms derived from a regional WAN RTT. In the case of a model using uniformly distributed interactivity time the maximum value is $m = 150$ ms and in the case of a shifted exponential distribution, $\mu = 20$.
- Extreme WAN VDI: An expected value of interactivity time of 600 ms with a minimum value of $r = 500$ ms. In the case of uniformly distributed interactivity time, the maximum value is $m = 700$ ms and for a shifted exponential distribution, we estimate a value $\mu = 10$.

In Fig. 5, one can see that for small values of interactivity time, all models provide similar results, whereas they differ for distributions of interactivity times with wider supports. We can expect an error rate of less than 5% based on instabilities when the inter-instability time is greater than 15 s, even in the extreme WAN VDI scenario.

Let $t_{s,i}$ be the duration of a stability measurement interval and T_s be the duration of the stability measurement interval required to achieve a certain stability error as obtained from Fig. 5. The stability check will flag each interactivity time sample according to the following rules:

- Stability measurement intervals with $t_{s,i} \geq T_s$ will flag the following interactivity measurements as reliable.
- Stability measurement intervals with $t_{s,i} < T_s$ will not change the flag for future interactivity measurements.
- The occurrence of an instability flags the following interactivity time as unreliable.

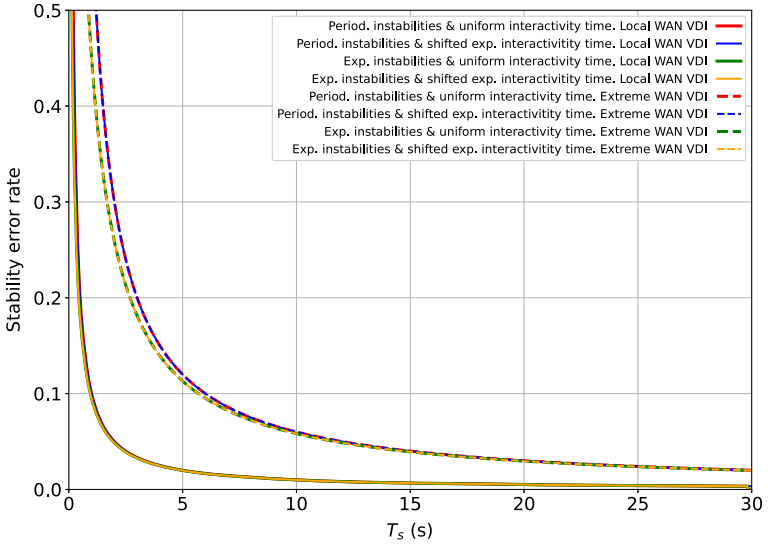

Fig. 5 Probability of measurement errors caused by instabilities

Figure 6 presents examples of instabilities, user actions, and stability measurement intervals to clarify the different cases and scenarios. Each stability measurement interval has a start time s_i and end time f_i, where $t_{s_i} = f_i - s_i$ and the start time s_i represents the time just after an interactivity time measurement or instability. The end time f_i represents the time just before the beginning of an interactivity time measurement or instability. Each interactivity time measurement has a user action e_i and screen update related to that action C_i, where the interactivity time is defined as $t_{int_i} = C_i - e_i$. Finally, instabilities I_i appear randomly. In the example, because $t_{s_1} \geq T_s$, the stability measurement interval t_{s_1} flags the following interactivity times t_{int_2} as reliable. Because $t_{s_3} < T_s$, there is no change in flagging, and t_{int_4} is flagged as reliable. When $t_{s_5} < T_s$, there is no change in flagging, but based on the instability occurrence at I_6, flagging is not reliable for future measurements. Because $t_{s_7} < T_s$, there is no change in flagging, and t_{int_8} is flagged as unreliable. Finally, when $t_{s_9} \geq T_s$ the flagging is changed, and $t_{int_{10}}$ is flagged as reliable.

Frequent stability measurement intervals longer than T_s are indicative of a very stable desktop and, therefore, of very reliable measures of interactivity time. We analysed a case of real users using office applications in a large company to determine how often such long stability measurement intervals occur. Measurements were performed over the workdays of a week in February of 2020. Figure 7 presents the CDF of the times between stability measurement intervals based on subsets of such intervals, which were conditioned to be longer than 1, 3, 5, 7, and 10 s. The results demonstrate that long stability periods are frequent; meaning large values of T_s still provide a substantial number of reliable measurements. For example, 90% of the stability measurement intervals longer than $T_s = 5\ s$ are less than 20 s apart. Therefore, in one minute we can expect to find two or three stability measurement intervals longer than 5 s, and the interactivity measurements immediately after these intervals will be marked as reliable.

It is convenient to evaluate desktop stability to notify users if conditions are optimal for measuring interactivity time. A user can decide to eliminate changing elements on the desktop that may affect measurements (e.g., removing desktop animation) or to perform an unreliable measurement. During the initial desktop calibration phase, the screen is divided into sectors based on input parameters and its stability is evaluated in these sectors. The presence of instabilities is reported in the screen section. During calibration, an average interactivity time is obtained and used to determine an objective minimum time for stability measurement intervals T_s using Eqs. (5), (7), (9), (11) or Fig. 5.

3.4 Measurement procedure

Figure 8 presents a block diagram of the proposed implementation of our methodology. Two main blocks are distinguished according to the phases of operation. First, when the tool is launched, it executes the 'initialisation block'. This block is called in the first moments of

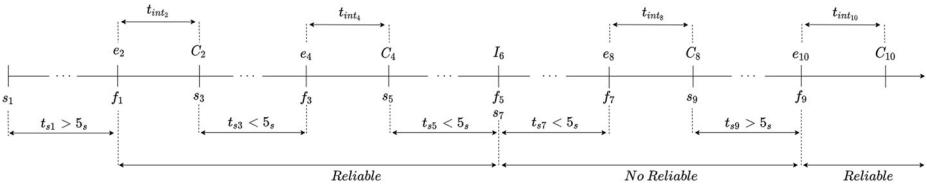

Fig. 6 Examples of instabilities, user actions, and stability measurement intervals

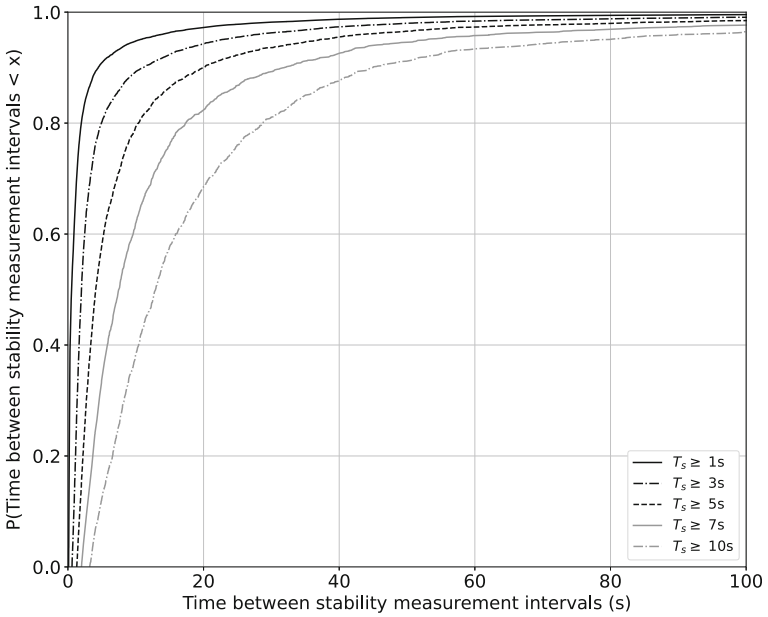

Fig. 7 Time between stability measurement intervals for real office users

running the program, and it fulfils the task of initialisation by requesting custom parameters from the user and preparing an environment for measurement extraction. Additionally, it performs desktop calibration with the goal of evaluating the variability of the user's environment. The custom parameters are the size of the evaluation area and whether the screenshots must be stored. In this phase, an estimation of the average interactivity time is obtained, and the system selects a target T_s threshold for the stability measurement interval.

The main loop begins by checking whether a change in mouse coordinates has occurred. A stability measurement is performed when the mouse coordinates are stable and no keyboard or mouse click user events are recorded. In the case of mouse movement during a stability measurement, measurement, ends and the following screen change is associated with the

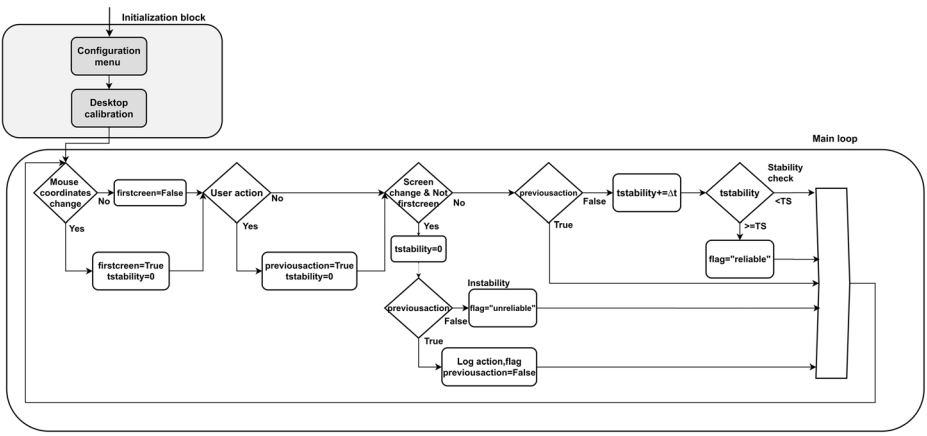

Fig. 8 Flowchart describing how interactivity time is measured, including the evaluation of instabilities

movement and discarded. Stability measurement also ends in the event of a keyboard or mouse click user action. In both cases, if the stability measurement interval is greater than or equal to T_s, the flag for interactivity time validation is updated to the value of reliable.

When the mouse coordinates are stable, a user action with the keyboard or a mouse button is registered with its timestamp (variable *previousaction* in Fig. 8), and an interactivity measurement is launched. In the case of mouse movement during this interactivity measurement (between the user action and screen change), the measurement is cancelled. The screen is checked for changes, and the first change after the user event is registered in the interactivity measurement. In the case of a screen change without a previous user action, the change is an instability and the following interactivity times are flagged as 'unreliable'.

Figure 9 presents a time diagram describing the manner in which measurement should be conducted. The monitor refresh rate is a fundamental aspect that conditions the elapsed time that users perceive between their interactions and screen updates. The thin client generates screen updates in S_i epochs, as shown in Fig. 9, at intervals of T_f (T_f = 16.6 ms for a refresh rate of 60 Hz). In this figure, each screen capture is denoted as C_n.

Interactivity time, which is defined as the time periods between a user's interactions and the screen captures of their associated responses C_{n+1}, is calculated as $t_{int} = t_{C_{n+1}} - t_{e_j}$, as shown in Fig. 9. The real interactivity time perceived by the user is associated with the screen update S_{i+1}. Therefore, $t_{error} = t_{C_{n+1}} - t_{S_{i+1}}$ represents the error in the measurement of interactivity time.

For time and space complexity reasons, the proposed methodology should limit the number of screen captures that are performed per screen refresh period T_f. In other words, the minimum time that can elapse between C_n and C_{n+1} should be limited. Therefore, it is necessary to obtain the thin client monitor refresh frequency and wait for a complete refresh period to be completed. This avoids evaluating the same frame twice.

The proposed methodology does not use the full screen size for the evaluation area. Therefore, responses that are represented outside the evaluation area may not be detected. If the tool does not consider this phenomenon, then the procedure will be blocked and no other measurements will be made until a new screen change is detected in the original evaluation area. This could lead to false negative measurements, meaning false measurements that provide worse results (longer measurements) than the actual interactivity time. Therefore, the detection of a new interaction (mouse or keyboard event) should interrupt the procedure and discard the previous interaction. This process continues until a new screen update is detected. To avoid losing two measures instead of just one, the evaluation area should be updated with the arrival of a new event. The discarded event is annotated jointly with its successor and the phenomenon is recorded.

Losing the actual time of a screen update can also occur if the algorithm does not run as quickly as required on a weak machine. Therefore, it is necessary to reduce the size of the

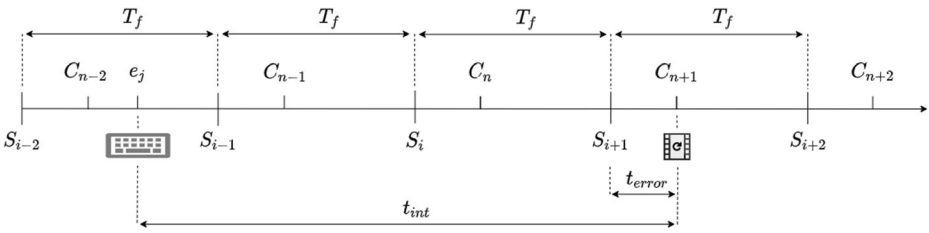

Fig. 9 Diagram of system events

evaluation area to decrease computation time. However, this can also occur if the thin client is grouping several user inputs together in the same network packet, but the remote desktop or VDI sends a single screen update associated with all inputs. This phenomenon can lead to the detection of a single response on the screen for two interactions. In this case, the correct way to proceed is to associate the two keyboard or mouse interactions jointly.

3.5 Optional storing of screenshots to disk

As mentioned previously, it can be worthwhile to provide contextual information for interactivity measures. Such contextual information can include bitmaps of a user's screen corresponding to the states before and after a keyboard or mouse interaction. These images can provide information regarding which service or application the user was using on the remote desktop at the time when an interactivity measure was extracted. This can help to discern if a problem at that timestamp was caused by a problem in the application, on the thin client, on the network, or on the server. The storage of images and additional contextual information (consumption of CPU power and memory) is considered to be optional, but it can be useful when evaluating measurements. Formatting a bitmap and writing it to a disk with a specific compression format has a significant cost. If such a screenshot will be used to examine the environment at the time of evaluation, a compressed image with loss that still allows for discernment of the environment can suffice. However, if machine-learning-based image processing will be performed, then the bitmap must be stored without loss to avoid compression artefacts. It is worth noting that this processing is performed after the extraction of interactivity time. The calculation of interactivity time is performed using a raw bitmap without any compression or formatting prior to computing its checksum.

Figure 10 presents the CDF of the time required to store one image with a resolution of 683 × 364 pixels depending on the image storage format (Table 1). This time is directly related to the type of hard disk used, amount of disk space required, and its usage. For our analysis, a hard disk with a DRAM buffer size of 8 MB, 3 Gb/s SATA connection interface, and data

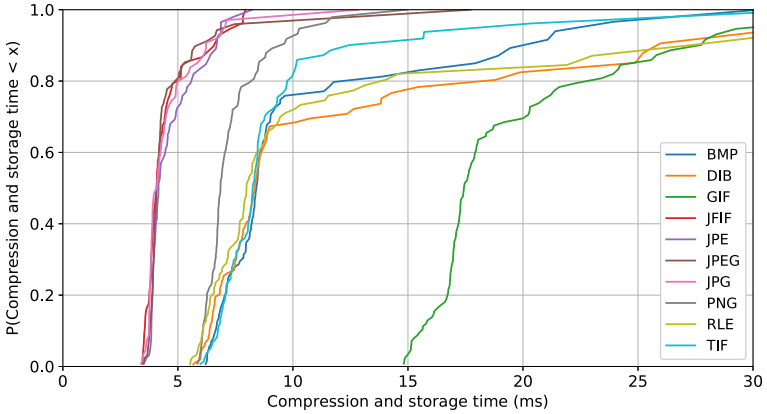

Fig. 10 CDF of processing and storage time required for one 683 × 364 pixel image depending on the storage and compression format

Table 1 Main characteristics of the formats used for analysis of the time required to process and store bitmaps [8, 18, 32, 36]

Image name	Extension	Compression	Colour Depth	Average compression ratio
Bitmap image file	BMP	Lossless: Optional - RLE	1–4 - 8 - 15 - 24 - 32 - 64	2:1
Device-Independent Bitmap	DIB	Lossless: Optional - RLE	4 for VGA bitmaps, and 8–16 - 24 - 32	2:1
Graphic Interchange Format	GIF	Lossless: Lempel–Ziv–Welch and an uncompressed version is feasible	1–4 - 8	2:1 to 5:1
JPEG File Interchange Format	JFIF/JPE/JPEG/JPG	Lossy: JPEG compression	8 or 24 (8 per channel)	16:1
Portable Network Graphics	PNG	Lossless: DEFLATE	1–4 - 8 - 16 - 32 - 64	13.9:1
The Utah RLE (Run Length Encoded) format	RLE	Lossless: RLE	8 bits per channel	2:1
Tagged Image File Format	TIF	Optional, may be lossless or lossy. Defined by TIF compression tag.	8 or 16 bits per channel	1.4:1 to 3.4:1

transfer rate of 300 Mb/s with an average latency of 5.6 ms was used. The main characteristics of the formats used for analysis are summarised in Table 1.

The shortest times are yielded by the formats that perform compression based on the ITU-T T.81 standard, which includes JPEG. There are several coding variants for images that share the same compression method. The aforementioned standard includes the JPEG, JPE, JFIF, and JPG formats, which, despite having different disk structures, behave similarly from a CPU cost perspective. JPEG compression is performed with loss dependent on the image quality configured in the compression process [22]. If subsequent image processing is required for captured images, it is of interest to use an algorithm that provides lossless compression. The format that meets these characteristics and has the shortest time cost following the JPEG compression family is the PNG format.

Therefore, if subsequent processing of captured images is not required, the best compression formats are JPEG and JFIF, which require a formatting and saving time for one image bounded between 4 and 8 ms. In contrast, if lossless compression is preferred, PNG requires a time for formatting and storing one image between 6 and 17 ms.

4 Experimental evaluation of TeCLA

The main accuracy limitations of the proposed tool are related to the screen refresh time, particularly the time interval referred to as t_{error} in Fig. 9. This interval represents the difference between the real interactivity time actually experienced by the user and the interactivity time captured by the implementation of the proposed methodology. To ensure the greatest precision of the obtained metric, this time interval should be as short as possible.

Analysis of the t_{error} values calculated by TeCLA was performed using two tools: a JitBit macro recorder [15], which emulates user interactions, and a custom program called an Interaction Delay Emulator (IDE) written in Python, which generates customised changes on the screen with a controlled delay when a mouse or keyboard interaction is detected. Both tools run on the thin client. The IDE can emulate a zero-delay infrastructure. Upon receiving a keyboard or mouse interaction, it prepares a change on the screen without any delay in its generation and evaluates the time required for representation (the time between the software sending the screen update to the operating system and the screen presenting the update). In this manner, we can measure the interactivity time that would be obtained in a local environment. TeCLA runs in parallel to perform interactivity time calculations. Therefore, t_{error} in Fig. 9 is defined by the difference between the measured interactivity time obtained by TeCLA and the exact time used by the IDE to update the screen when it receives an interaction. Tests were conducted using a computer with two CPU cores running at 2.0 GHz, 8 GB of RAM, a Windows 10 Home installation (10.0.17134), and a monitor with a 60 Hz refresh rate and screen resolution of 1366 × 768 pixels. Measurements were extracted with a nanosecond resolution.

As stated previously, the refresh rate of the monitor is a fundamental constraint on the behaviour of the proposed tool. The thin client will not try to represent new information on the screen until a full refresh cycle has been completed. The program will extract the first checksum in a refresh period T_f and wait until a full cycle is completed, allowing it avoid evaluating the same frame twice. This procedure implies that the tool will begin its evaluation at a timestamp within the period of a screen refresh, which will determine the accuracy of measurement. It should be noted that the graphics card also influences measurement accuracy. This is because when the IDE detects a keyboard or mouse interaction, it will attempt to present a change on the screen as quickly as possible. However, because the instruction to modify the status of the graphics card is delayed until it can be executed, in the worst case, a full refresh period can elapse. If the time at which the monitor is updated is not controlled, then the t_{error} measurements will be altered, resulting in a greater delay. To assess when the IDE presents an on-screen update accurately, the Python PsychoPy library [27] was adopted. This library was designed to perform studies in the fields of psychophysics, cognitive neuroscience, and experimental psychology. PsychoPy allows visual stimuli to be generated in a controlled manner. Specifically, it facilitates the calculation of the time elapsed from a draw-on-screen instruction being issued to the time when the corresponding representation is visible on a user's monitor. In this manner, it is possible to evaluate the real error achieved by TeCLA.

For accurate analysis, the parameters specified in Section 3 were adopted. The results of analysis are presented for evaluation area sizes of sectors of the full screen (1366 × 768 pixels) split in half (683 × 364 pixels), in fourths (342 × 182 pixels), and in eighths (171 × 91 pixels). The most convenient setting for colour depth is 16 bits, so the results presented below were obtained for this colouration configuration.

Figure 11 presents the CDF of the accuracy (t_{error}) obtained by the proposed tool for different dimensions of the evaluation area according to the accuracy evaluation strategy discussed above. If the evaluation area increases, the deviation between the real value and the measurement increases based on the extra cost of screen captures. The CDF for the largest area (683 × 364 pixels) exhibits values between 2.6 and 4.5 ms. For the other scenarios, the precision provided by TeCLA varies proportionally to the evaluation area size. In the case of 342 × 182 pixels, t_{error} is between 1.6 and 3.8 ms, while in the case of the least demanding evaluation area size (171 × 91 pixels), values between 0.2 and 1.6 ms can be observed. One

can see that the delay remains significantly below the aforementioned bottleneck of the monitor screen refresh rate, allowing the extraction of interactivity times close to the theoretical limit. These times are significantly less than the typical RTTs observed in WAN VDI deployments [21].

As discussed above and shown in Fig. 2, the time required to extract a bitmap and process its checksum depends on the evaluation area size. The time spent on this process should not be considered when t_{error} is calculated because the time reference must be recorded at the time when an image is extracted, rather than when it has been processed. However, it is possible to appreciate how the influence of the evaluation area size becomes perceptible in the CDF of t_{error}. This is because the order of magnitude of the time required to extract a bitmap determines the time at which the thin client's screen can be processed.

It is important to minimise the resources involved in the proposed implementation. One of the main advantages of using VDI solutions is the reduction in costs associated with the management of user resources. Typically, the thin clients used to access infrastructures do not have significant computational resources because the corresponding features are outsourced by relegating processing to the server side. The proposed solution should seek to minimise its impact on a client's equipment by reducing its usage of the CPU and RAM as much as possible. This can also have an important impact on the performance of our tool. The average RAM and CPU usages of our tool were compared based on the storage of previous and subsequent bitmaps corresponding to user interactions. The results are presented in Table 2. The RAM figures correspond to the private bytes reserved specifically by TeCLA that cannot be accessed by other processes. Analysis was conducted for four possible configurations of the evaluation area. First, we present the necessary resources for full-screen configuration. The results of analysis are also presented for evaluation area sizes corresponding to the screen division ratios of two, four, and eight provided by the proposed implementation.

Table 2 reveals that RAM and CPU usage are low, particularly for small evaluation areas, allowing TeCLA to run on thin clients in real time. Additionally, if the tool is launched without storing screenshots, then resource usage is improved slightly. This table also reveals the manner in which the dimensions of the evaluation area affect the impact of TeCLA on the client's equipment, with larger evaluation areas being more demanding. Our tool requires an average CPU usage of 9.97% in the worst-case scenario. This corresponds to the process of storing screenshots of the former and

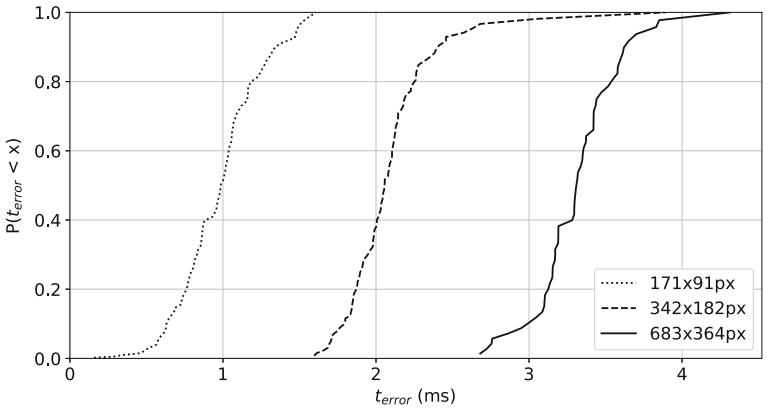

Fig. 11 CDF of t_{error} for a 60-Hz monitor for different sizes of the evaluation area

Table 2 Resource consumption of our tool with and without storing screenshots to the disk for different sizes of evaluation areas

Evaluation area	Storage enabled		Storage disabled	
	(RAM)	(CPU)	(RAM)	(CPU)
1336×768 pixels	7.8 MB	9.97%	6.4 MB	8.34%
683×364 pixels	6.25 MB	4.8%	5.2 MB	4.6%
342×182 pixels	5.2 MB	1.9%	4.8 MB	1.7%
171×91 pixels	5 MB	1.1%	4.7 MB	0.9%

latter states of a user interaction with the highest evaluation area size configuration of 1336 × 768 pixels. The lowest CPU usage is 0.9%, which is obtained for the largest sector ratio considered in this study without storing bitmaps. In terms of RAM usage, the most demanding option (7.9 MB) also corresponds to the configuration that employs the largest evaluation area and stores raster graphics of interaction states. Additionally, the selection of the smallest dimensions for the evaluation area and the use of the non-stored bitmap option represent the most efficient configuration, requiring only 4.7 MB of RAM. This table demonstrates that when the evaluation area does not correspond to the full screen resolution, the required resources are relatively small.

The infrastructure analysis performed by Buzen [6] indicated that if the CPU resources of a client or server machine are not being used at high levels, it can be concluded that the system behaviour is not being conditioned by power shortages for devices. The greater the amount of free CPU power, the lower the dependency between processes on the machine. Therefore, the results in Table 2 indicate that the proposed interactivity measurement procedure does not significantly affect the extracted results.

5 Use cases and experimental results

To demonstrate the feasibility and effectiveness of the proposed methodology, two desktop virtualisation infrastructures were evaluated. One was the VDI deployment at our university, which is intended to serve 850 concurrent remote desktops with the VDI servers in the same campus network. The other infrastructure is Amazon Workspaces, where the VDI server resides on the public cloud provided by Amazon. We refer these deployments as local and public VDI deployments, respectively.

Although the proposed methodology is designed to evaluate the interactivity of an infrastructure while users perform normal activities, to evaluate both the local and public VDI deployments, the process of interaction with these infrastructures was automated. To this end, a virtual machine instance was launched on the target analysis infrastructure. Within the virtualised machine, the IDE program was used to generate changes on the screen for each user interaction. It should be noted that IDE does not introduce any extra delay. The JitBit macro recorder was used to emulate mouse events on the remote desktop. TeCLA was launched on the thin client to evaluate the interactivity time between the interaction events emulated by JitBit and the controlled responses generated by the IDE.

This study was conducted using thin clients on a campus network, which were used to monitor both the local and public VDI infrastructures continuously. The system configuration

was identical for all three hosts with two CPU cores running at 2.66 GHz, 4 GB of RAM, and an installation of Microsoft Windows 10.

5.1 Local VDI

The local VDI infrastructure at our university allows users to select the type of instance they wish to use. It is possible to select an instance with a Microsoft Windows 10 Server 2016 Data Center installation with a CPU running at 2.4 GHz and 4 GB of RAM. There is also a Linux-based option with an installation of Ubuntu 18.04.093 with the same processor frequency and RAM. For both systems, the screen refresh rate corresponds to the local host used for testing (60 Hz) with a screen resolution of 1552×846 pixels. According to the results presented in Section 3.1, an evaluation area of 572×423 pixels (one quarter of the screen area) and a setting of 16 bits for colour depth information were used. The software used by clients to access the remote desktops at the university varies depending on the operating system of the chosen instance. The RDP client provided by Microsoft is used for the Microsoft Windows option. However, to use the Linux-based desktop, users must use a web browser.

Over the course of our experiments, analysis of the available bandwidth and the latency experienced by the equipment within the university network was performed using the IPerf tool [11]. The average transfer rates were approximately 92 Mb/s downstream and 94.6 Mb/s upstream. These rates are limited by the 100Base-T connectivity of the computers, but are still much faster than the minimum speeds required needed for remote desktop usage [20]. The average measured RTT was 7 ms, which is far below the 150 ms recommendation of ITU-T G.114 standard [13]. Based on these network metrics, our interactivity measurements should not be biased by the influence of the access network.

The interactivity time of the VDI infrastructure was measured using TeCLA from April 4, 2019 to May 28, 2019, representing more than a month and a half of continuous monitoring, to study the behaviour of the system, including both types of instances that can be launched in the VDI. This study revealed the interactivity time that the infrastructure is capable of providing for each type of instance. Figures 12 and 13 present the measurements obtained throughout the week of April 29, 2019. The CDF of the interactivity time experienced by users is represented per day of the week. Almost all of the samples obtained for the Windows instances are below 150 ms and above 50 ms. In contrast, the virtualised Linux-type instances reflect higher variability and higher interactivity times. Most of the measurements obtained for the Linux instances are below 500 ms and above 125 ms. These results show that the target service offers better interactivity for sessions that are launched using the Windows operating system option.

One can observe that the samples obtained for Windows-type instances tend to be distributed more homogeneously. In contrast, the measures extracted for the Linux-type instances reflect a trend of clustering at approximately 200 ms and 500 ms. The measurements evaluated for the Linux option are below 200 ms approximately 80% of the time, whereas the analysis performed on Windows-based instances reveals values below 100 ms approximately 80% of the time. The network and servers are the same for both instances, so the main differences lie in the optimisation of the VDI client running on the thin clients and the remote desktop protocol used. Windows uses the Microsoft Remote Desktop protocol [26], whereas Linux uses XRDP [38]. These protocols have a significant effect perceived interactivity.

Our monitoring also detected increase in the interactivity time experienced by infrastructure users at certain times of the day. Figure 14 presents the behaviour of the system for a Windows-type instance on April 10, 2019. This figure represents average samples at 10 s

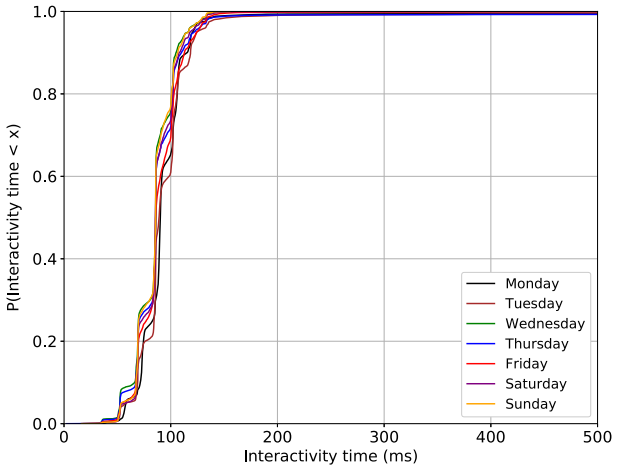

Fig. 12 CDF of the interactivity time of Windows instances on the local VDI over one week

intervals. One can see an increase at approximately 13:08, where there are average delays of up to 300 ms. This behaviour is repeated to a lesser extent at approximately 13:50 with an average interactivity time of 150 ms. One can also observe specific moments with short durations (10 s) in which interactivity is impaired to a greater extent, reaching average delays of up to 450 ms. This behaviour occurs at approximately 14:35 and, to a lesser extent, at approximately 15:40. The cause of these peaks in interactivity time was confirmed by the service administrator. The peaks can be attributed to saturation of the infrastructure with an excessive number of instances running. This occurs during some university class changes when some students do not close their instances, but only close the thin client software, while new students open new instances. This leads to an overload on the cluster of servers.

Interactivity time remains controlled and constant most of the time, but TeCLA facilitates the detection of the moments in which the interactivity perceived by users declines. Additionally, in both Figs. 12 and 13, one can observe a series of steps in the CDF that reveal a tendency of the samples obtained to be grouped around values close to multiples of the refresh

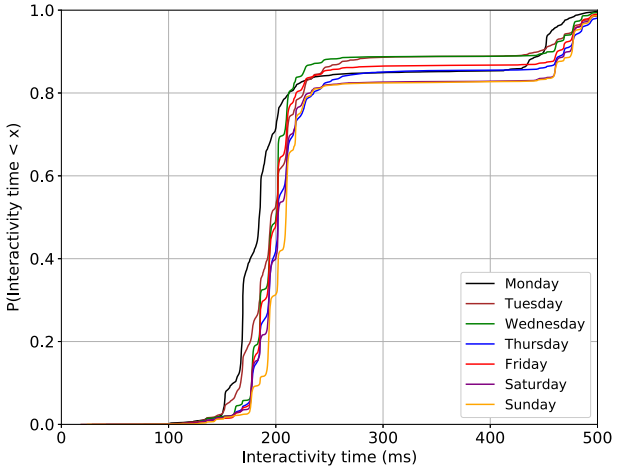

Fig. 13 CDF of the interactivity time of Linux instances on the local VDI over one week

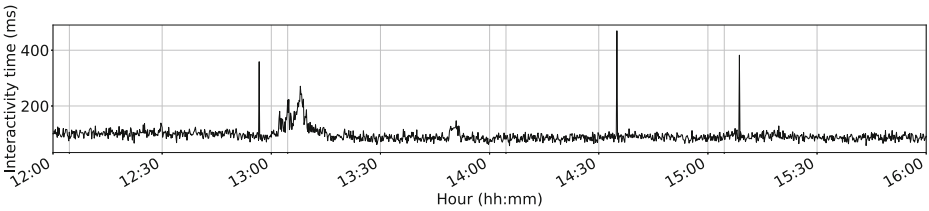

Fig. 14 Measures obtained for a Windows-type instance on April 10, 2019, with the average calculated every 10 s

period of the screen used for analysis (16.6 ms). These values correspond to the same trend that was previously illustrated in Fig. 2, meaning the shape of the CDF of the interactivity time experienced by users is conditioned by the display frequency of the monitor used by a thin client.

Linux-based desktop users must use a web browser to access a remote desktop on the local VDI deployment. The influence of the type of browser on the interactivity time experienced by Linux users was analysed. The analysed browsers were Firefox 68 and Google Chrome 73.0.3683. Both tests were performed simultaneously. Figure 15 presents the CDFs of the interactivity times experienced for both browsers. One can observe the trend mentioned previously for the Linux-type instances, which exhibit interactivity times grouped at approximately 200 and 500 ms. Additionally, the shapes of the distributions are similar for both browsers. However, our analysis reveals better interactivity times for Firefox. One can see that 80% of the Firefox measurements have interactivity time values lower than 212 ms, while in the case of Google Chrome, this value is 25 ms higher.

5.2 Public VDI: Amazon WorkSpaces

The remote desktop virtualisation infrastructure of Amazon WorkSpaces was also monitored using TeCLA. It was of interest to analyse imbalances in the interactivity perceived by users when the VDI resides within the same network versus when a public network must be used to access the VDI, as well as the different server capabilities and protocols involved. This experiment was conducted on our campus between May 2 and May 28, 2019. The results discussed below were extracted using a screen resolution of 1552 × 846 pixels and a refresh

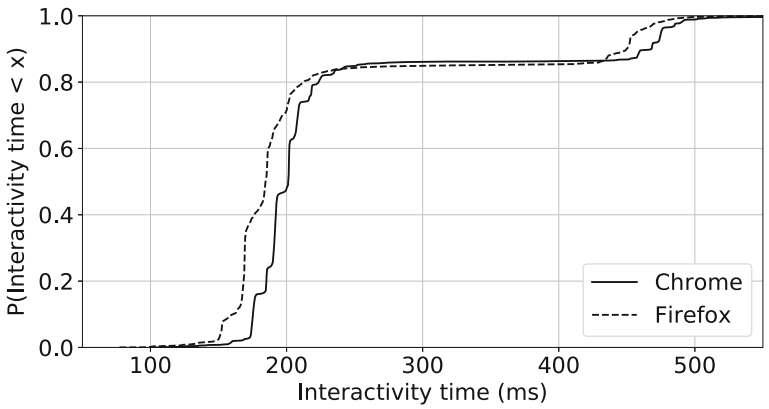

Fig. 15 CDFs of the perceived interactivity time depending on the browser used in the Local VDI deployment

rate of 60 Hz. Additionally, according to the results presented in Section 3.1, an evaluation area of 572 × 423 pixels and setting of 16 bits for colour depth information were adopted.

The Amazon WorkSpaces virtual desktop infrastructure allows users to select the type of instance they wish to use. We configured an environment with a Windows 10 installation, one virtual CPU, 2 GB of RAM, and 10 GB of disk space per user. All tests were performed by using the client software provided by Amazon with its last available version (2.5.6.1183) to access the service. Most of this study was conducted using the Amazon servers closest to our campus, which are located in Ireland. During the course of this experiment, analysis of the available bandwidth and latency experienced by the thin client for a server launched on the Ireland site was conducted using the IPerf tool. Average transfer rates of approximately 20.6 Mb/s downstream and 27.4 Mb/s upstream were recorded. Some papers in the literature have addressed the characterisation of the network requirements for the PCoIP protocol used by Amazon WorkSpaces. In [20, 39], it was demonstrated that the video user profile tends to be the most demanding service, requiring a transfer rate of 0.834 Mb/s in the upstream direction and 9 Mb/s in the downstream direction. Therefore, the bandwidth available for our setup is not a bottleneck. The average measured RTT was 65 ms. Even if this value can be considered as a large RTT value, according to the ITU-T G.114 recommendation, there is a suitable margin for the processing delay at the VDI platform.

Figure 16 presents the CDF of the interactivity time extracted during the week of May 13, 2019 for a virtualised Windows-type instance in the Amazon WorkSpaces service located in Ireland. Almost all of the samples obtained lie between 150 and 400 ms. One can see that the variation in interactivity time over the evaluated days is small. The measurements are distributed in a similar manner and are homogeneous to a certain degree. Approximately 80% of the interactivity times are below 350 ms. The extracted results reveal the influence of the WAN network on the interactivity provided by virtual desktop virtualisation infrastructures. The study on the local VDI revealed that in the worst case (virtualisation of Linux-type environments), 80% of the measurements were below 200 ms. In contrast, only 10% of the samples obtained for Amazon WorkSpaces lie below that value. Again, the shape of the cumulative time-of-interactivity distribution function presented in Fig. 16 exhibits a series of

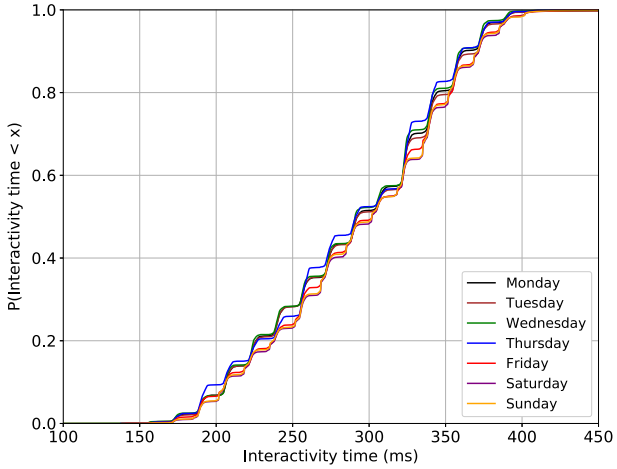

Fig. 16 CDF of the interactivity time for Amazon Workspaces over one week

steps that reveal the tendency of the analysis samples to be grouped around the refresh period corresponding to the 60 Hz monitor frequency used for analysis.

Additionally, the influence of the location of the Amazon Workspace instance on the interactivity times experienced by users was analysed. For this purpose, a VDI instance was configured with the same characteristics as the VDI instance launched in Ireland, but it was launched in different locations. We tested locations in the United States (North Virginia and Oregon) and Pacific Asia (Sydney). These locations were selected because when our experiments were conducted, they were within the selectable options in the main regions where Amazon Web Services are available.

Based on changes in the location of the instances, another analysis of the network access conditions for thin clients located on the campus network was performed. The average RTTs measured for the different locations were 119 ms for North Virginia, 181 ms for Oregon, and 381 ms for Sidney. Figure 17 presents the CDFs of the interactivity times tested for different locations of the Amazon WorkSpaces service for one day during the week of May 21, 2019. An imbalance in interactivity time can be observed based on the instance location. One can see that 80% of the measurements extracted for North Virginia are below 370 ms, while almost of the remaining measurements are below 500 ms. For the Oregon location, 80% of the samples obtained are below 459 ms and the remaining 20% are below 550 ms. For the Sydney location, 80% of the measurements are below 638 ms, while the other 20% are below 700 ms. For the North Virginia location, the average RTT is 54 ms less than that for the Irish location. However, the differences in interactivity times between Ireland and North Virgina do not correlate with the statistics summarised above, where one would expect the values to be significantly worse for the North Virginia location compared to the Ireland location.

Regardless, the other locations exhibit interactivity time differences in line with the average RTTs measured during testing. To delve more deeply into this phenomenon, another server was launched on the North Virginia Amazon Web Services platform to determine if the access network could be biasing measures. In this case, the experiment revealed a 16.4 Mb/s upstream bandwidth and 9.58 Mb/s downstream bandwidth. Additionally, the transfer rate required for the VDI network traffic was measured, resulting in a maximum downstream peak of 2.27 Mb/s and maximum upstream peak of 1.26 Mb/s.

In Fig. 18, one can see how the transfer rates for all locations except North Virginia exhibit a downstream throughput close to 0.220 Mb/s. However, the instance located in the eastern

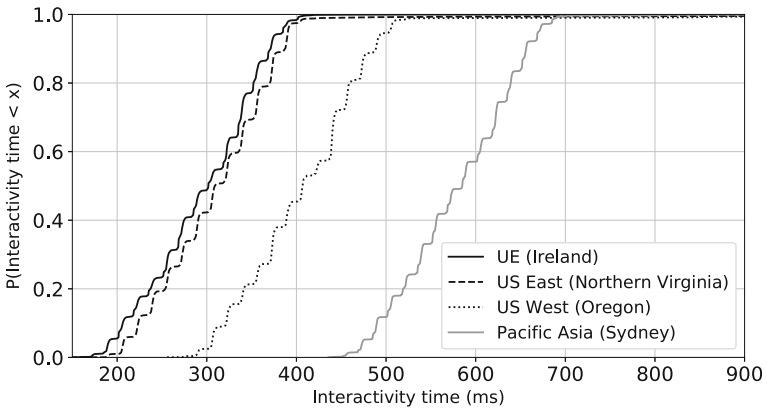

Fig. 17 CDFs of interactivity times for different VDI server locations

United States seems to be performing greater compression based on the fact that the information sent to the client is sent at approximately 0.180 Mb/s. These results do not exhibit significant variance, but the reduced use of bandwidth occurs exclusively for this location. The processing required for compression by the servers could be partially influencing the interactivity perceived by the thin client. The bit rate required to transport the changes performed by screen updates can be reduced by using stronger compression. Coarse compression can reduce the information required in the transmission, but can worsen the visual quality of the connection in favour of reduced delay. Additionally, ad-hoc hardware can also be used on servers to provide image compression with less compression delay. The measures obtained from the North Virginia VDI analysis should be biased by the large RTT compared to the Ireland server, but they are not. Additionally, during analysis, there was a slight degradation in the visual quality of the remote desktop located in Northern Virginia. Therefore, in Northern Virginia, the loss of image quality caused by a high compression ratio and superior compression hardware yields better interactivity time than expected based on the corresponding RTT.

6 Related work

There are several works in the literature that have attempted to address the study of user experience for remote desktop environments using various approaches. A large number of researchers base their studies on the Slow Motion Benchmarking [24] technique, which was proposed by Jae Yang to monitoring the packets exchanged between a client and server. Several researchers have attempted to identify the packets sent from a client as user interactions and packets received by a client as screen updates. However, in this method, it is necessary to introduce delays into user interactions (instrumentalising) to ensure that changes are represented on the client before the next user interaction is sent. This alters the normal behaviour of the applications and solutions used in many services, thereby hampering practical interactivity evaluation. Our proposed methodology allows us to obtain the interactivity times experienced by real users in a real environment by studying how they behave within their own environments, regardless of the applications they are using or the activities performed. Additionally, the Slow Motion Benchmarking technique does not account for click-ahead behaviour, meaning it is not considered that a user can interact with a thin client before any information has been updated as a response to a previous interaction. Additionally, a thin client

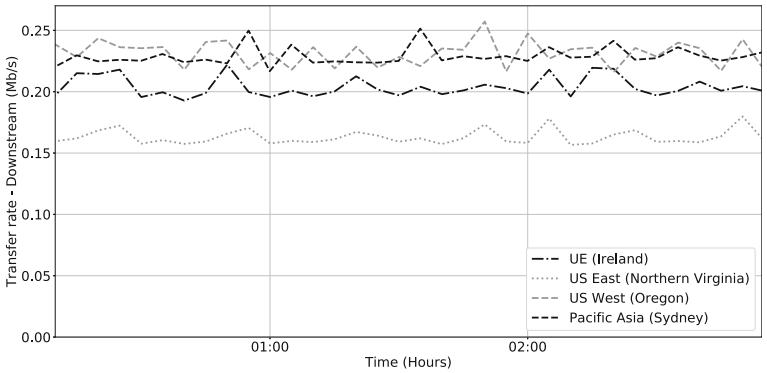

Fig. 18 Time series of downstream throughput for each VDI server location (5 min intervals)

may group multiple user inputs into a single network message, and then the remote desktop or VDI may send a single screen update associated with multiple inputs. The methodology proposed in our work groups interactions in the presence of this phenomenon.

It is difficult to generalise an implementation based on network traffic because such an implementation requires the dissection of the underlying protocol. Some studies have implemented their proposals focusing on a specific protocol, such as [21], which focused analysis on the RDP protocol. Nevertheless, in our case, the processing is carried out in real-time and allows the evaluation of different remote desktop solutions independently of the protocol used, supporting cyphered protocols and without making modifications in its implementation.

Berryman et al. [5] proposed a solution called VDBench. They defined a series of procedures and metrics to allow the evaluation of thin clients, focusing exclusively on RDP [26], RGS [25] and RealVNC [29]. They used the Slow Motion Benchmarking technique to study the relationship between the number of virtualized instances in a centralized environment and the quality perceived by the user. However, the measurements were performed on the server, so they did not reflect the client's perspective.

Other studies have attempted to evaluate the client's experience empirically, such as [31], in which the authors sought to relate network parameters to the quality of service offered by the remote desktop. The problem with this proposal is that there is not a direct mapping between the service and the network metrics. Conversely, Hongdi Zheng et al. [40] used the time between screen updates to evaluate the QoE of users in different transmission schemes for desktop virtualization environments, with a low frequency being indicative of a worse user experience. However, the time between screen updates is not compared with user's actions within the system.

Other lines of research, such as [6], have used the ITU-T G.1030 recommendation [14] to support the analyses. They have used RTT measures and maximum session times to obtain MOS values to quantify how well a service is running. The problem with this approach is that the best samples are expressed by an MOS value of five, but those samples can have excessive response times for clients, reducing reproducibility and making the comparison of results difficult. In [7, 16], the authors attempted to analyse the response times that a user experiences between interactions and screen update representations. Screenshots were captured continuously to determine the time spent on monitor refresh cycles and subsequent processing prior to the registration of user actions. However, they did not specify all of the procedures involved in their study process. Therefore, the consequences and implications of their proposed monitoring process were not analysed and their approach did not allow real-time evaluation. Therefore, the usability and feasibility of their approach were subject to the time required for analysis because their method requires variable time for subsequent processing. Additionally, the time required for a response to be received, processed, and represented, which we defined in this paper, has not been taken considered in previous studies. We also considered real-world infrastructures and distinguished virtualised equipment between local and public VDI solutions.

7 Conclusions and future work

We proposed a novel measurement technique called TeCLA for the quantification and analysis of the interactivity experienced by users of remote desktop and cloud applications. The proposed measurement technique is based on the calculation of the time required for the process of sending user interactions, processing them, and representing the associated

responses on the client's monitor. This approach requires extracting raster graphics of a thin client's screen and calculating their checksums for fast comparisons to earlier screen states. This implementation facilitates the detection of screen updates in real time.

The feasibility of our implementation was analysed, and measurements were obtained with a precision distributed between 2.5 and 4.5 ms in the worst-case scenario. This scenario corresponds to an evaluation area of 683 × 364 pixels and a monitor update period of 16.6 ms for a 60 Hz monitor. Our measurement procedure is auto-calibrated to detect instabilities in the screen and reduce the rate of erroneous measurements to below a specified threshold. We demonstrated that a 95% proportion of valid measurements can be obtained in a real scenario.

We evaluated the interactivity times perceived by the users of the remote desktop virtualisation infrastructures in a local VDI deployment and a cloud solution accessible from a WAN environment (Amazon WorkSpaces). The results highlighted the influences of the protocols, client optimisations, server resources, and VDI location on the experience provided.

In future work, we plan to perform automatic processing of extracted images combined with record logging and to use machine learning techniques to differentiate anomalous VDI behaviours from application behaviours.

Code availability Not applicable.

Authors' contributions

- **Conceptualization**: Jesus Arellano-Uson, Eduardo Magaña.
- **Data curation**: Jesus Arellano-Uson, Eduardo Magaña.
- **Formal analysis**:Jesus Arellano-Uson, Eduardo Magaña.
- **Funding acquisition**: Eduardo Magaña, Daniel Morató, Mikel Izal.
- **Investigation**: Jesus Arellano-Uson, Eduardo Magaña.
- **Methodology**: Jesus Arellano-Uson, Eduardo Magaña, Daniel Morató, Mikel Izal.
- **Project administration**: Eduardo Magaña.
- **Resources:** Eduardo Magaña.
- **Software**: Jesus Arellano-Uson.
- **Supervision**: Eduardo Magaña.
- **Validation:** Eduardo Magaña,Daniel Morató, Mikel Izal.
- **Visualization**: Jesus Arellano-Uson, Eduardo Magaña.
- **Writing– original draft**: Jesus Arellano-Uson, Eduardo Magaña.
- **Writing– review & editing**: Jesus Arellano-Uson, Eduardo Magaña, Daniel Morató, Mikel Izal.

Funding This work was supported by Spanish State Research Agency, project number PID2019-104451RB-C22/AEI/https://doi.org/10.13039/501100011033.

Data availability Not applicable.

Declarations

Conflicts of interest/competing interests Not applicable.

References

1. Amazon Web Services (2020) Overview of Amazon Web Services [White Paper]
2. Amazon WorkSpaces website (2020). https://aws.amazon.com/es/workspaces/. Accessed 18 Mar 2020
3. Apache Guacamole website (2020). http://guacamole.apache.org/. Accessed 18 Mar 2020
4. AutoHotkey website (2020). https://www.autohotkey.com/. Accessed 18 Mar 2020
5. Berryman A, Calyam P, Honigford M, Lai AM (2010) VDBench: a benchmarking toolkit for thin-client based virtual desktop environments. In: 2010 IEEE second international conference on cloud computing technology and science. IEEE, pp 480–487
6. Buzen JP (1976) Fundamental operational laws of computer system performance. Acta Informatica 7:167–182. https://doi.org/10.1007/BF00265769
7. Casas P, Seufert M, Egger S, Schatz S (2013) Quality of experience in remote virtual desktop services. Pp 1352–1357
8. Fidler A, Likar B, Pernus F, Skaleric U (2002) Comparative evaluation of JPEG and JPEG2000 compression in quantitative digital subtraction radiography. Dento-Maxillo-Facial Radiol 31:379–384. https://doi.org/10.1038/sj.dmfr.4600724
9. Google Remote Desktop (2021). https://remotedesktop.google.com. Accessed 23 Mar 2021
10. Google Stadia (2020). https://store.google.com/magazine/stadia. Accessed 18 Mar 2020
11. IPerf website (2020). https://iperf.fr/. Accessed 18 Mar 2020
12. ITU-T (2002) Error-Correcting Procedures for DCEs Using Asynchronous-to-Synchronous Conversion V.42 (03/02).
13. ITU-T (2003) One-way transmission time G.114 (05/03).
14. ITU-T (2014) Estimating end-to-end performance in IP networks for data applications G.1030(02/2014).
15. JitBit Macro Recorder website (2020). https://www.jitbit.com/macro-recorder/. Accessed 18 Mar 2020
16. Rhee J, Kochut A, Beaty K (2009) DeskBench: flexible virtual desktop benchmarking toolkit. In: 2009 IFIP/IEEE international symposium on integrated network management. IEEE, pp 622–629. https://doi.org/10.1109/INM.2009.5188870
17. Kleinrock L (1975) Queueing systems: volume I – theory. Wiley Interscience. ISBN 978-0471491101. Pp.172, eq 5.10
18. Kodituwakku S, U SA (2010) Comparison of lossless data compression algorithms for text data. Indian J Comput Sci Eng 1:416–425
19. LapLink website (2021). https://web.laplink.com/. Accessed 23 Mar 2021
20. Magaña E, Sesma I, Morató D, Izal M (2019) Remote access protocols for desktop-as-a-service solutions. PLoS One 14:e0207512. https://doi.org/10.1371/journal.pone.0207512
21. Michael R, Shimba F (2012) A critical performance analysis of thin client platforms. In: 2012 second international conference on digital information and communication technology and its applications (DICTAP). IEEE, pp 383–387
22. Ngo QT, Md Abu L, Pham XQ et al (2017) A remote display QoE improvement scheme for interactive applications in low network bandwidth environment. Multimed Tools Appl 76:22217–22241. https://doi.org/10.1007/s11042-017-4692-z
23. Nieh J, Yang SJ, Novik N (2000) A comparison of thin-client computing architectures. Columbia Univ. https://doi.org/10.7916/d8z329vf
24. Nieh J, Yang SJ, Novik N (2003) Measuring thin-client performance using slow-motion benchmarking. ACM Trans Comput Syst 21:87–115. https://doi.org/10.1145/592637.592640
25. Packard H (2006) HP Remote Graphics Software [White paper].
26. Partridge C, Hinden R (1990) Version 2 of the reliable data protocol (RDP). RFC 1151
27. Peirce J, Gray JR, Simpson S, MacAskill M, Höchenberger R, Sogo H, Kastman E, Lindeløv JK (2019) PsychoPy2: experiments in behavior made easy. Behav Res Methods 51:195–203. https://doi.org/10.3758/s13428-018-01193-y
28. Peterson W, Brown D (1961) Cyclic codes for error detection. Proc IRE 49:228–235. https://doi.org/10.1109/JRPROC.1961.287814
29. RealVNC website (2020). https://www.realvnc.com/es/connect/download/viewer/. Accessed 18 Mar 2020
30. Richardson T, Levine J, RealVNC Ltd. (2011) The Remote Framebuffer Protocol.

31. Saiz E, Ibarrola E, Cristobo L, Taboada I (2014) A cloud platform for QoE evaluation: QoXcloud. In: proceedings of the 2014 ITU kaleidoscope academic conference: living in a converged world - impossible without standards? IEEE, pp 241–247. https://doi.org/10.1109/Kaleidoscope.2014.6858471
32. Scarmana G (2014) Lossless data compression of grid-based digital elevation models: a png image format evaluation. ISPRS Ann Photogramm Remote Sens Spat Inf Sci II–5:313–319. https://doi.org/10.5194/isprsannals-II-5-313-2014
33. Su X, Wu M, Xu J (2014) A novel virtual storage area network solution for virtual desktop infrastructure. In: 2014 international symposium on wireless personal multimedia communications (WPMC). IEEE, Sydney, pp 204–208
34. TeamViewer website (2020). https://www.teamviewer.com. Accessed 18 Mar 2020
35. Teradici What is PCoIP Technology (2021). https://www.teradici.com/pcoip-technology/what-is-pcoip. Accessed 24 Mar 2021
36. Welch T (1984) A technique for high-performance data compression. IEEE Trans Comput 17:8–19. https://doi.org/10.1109/MC.1984.1659158
37. Windows Remote desktop client website (2021). https://docs.microsoft.com/en-us/windows-server/remote/remote-desktop-services/clients/remote-desktop-clients. Accessed 24 Mar 2021
38. XRDP website (2020). http://xrdp.org/. Accessed 18 Mar 2020
39. Yuan F (2019) Computer desktop visualization image compression based on fuzzy clustering algorithm. Multimed Tools Appl. https://doi.org/10.1007/s11042-019-7241-0
40. Zheng H, Liu D, Wang J, Liang J (2019) A QoE-perceived screen updates transmission scheme in desktop virtualization environment. Multimed Tools Appl 78:16755–16781. https://doi.org/10.1007/s11042-018-7058-2