On the Internet-scale Streaming of Holographic-type Content with Assured User Quality of Experiences

Ioannis Selinis[1], Ning Wang[1], Bin Da[2], Delei Yu,[2] and Rahim Tafazolli[1]

[1]Institute for Communication Systems 5GIC, University of Surrey, Guildford, United Kingdom
[1]{ioannis.selinis, n.wang, r.tafazolli}@surrey.ac.uk
[2]{dabin, yudelei}@huawei.com
[2]Network Technology Laboratory, Huawei Technologies

***Abstract*—Holographic-type Communication (HTC) has been widely deemed as an emerging type of augmented reality (AR) media which offers Internet users deeply immersive experiences. In contrast to the traditional video content transmissions, the characteristics and network requirements of HTC have been much less studied in the literature. Due to the high bandwidth requirements and various limitations of today's HTC platforms, large-scale HTC streaming has never been systematically attempted and comprehensively evaluated till now. In this paper, we introduce a novel HTC based teleportation platform leveraging cloud-based remote production functions, also supported with newly proposed adaptive frame buffering and end-to-end signalling techniques against network uncertainties, which for the first time is able to provide assured user experiences at the public Internet scale. According to our real-life experiments based on strategically deployed cloud sites for remote production functions, we have demonstrated the feasibility of supporting user assured performances for such applications at the global Internet scale.**

***Index Terms*—Holographic-type Communication, Augmented Reality, Remote Production, Quality of Experience (QoE).**

I. Introduction

Holographic-type Communication (HTC) [1] is a type of new immersive media that combines both the Augmented and Virtual Reality (AR/VR) technologies to display in full 3D objects captured by RGB depth (RGB-D) sensor cameras. It is anticipated that in the near future, HTC based teleportation will become an increasingly popular over-the-top (OTT) application that allows Internet users to communicate with more immersive experiences compared to traditional video based applications. Moving from 2D to 3D objects would significantly increase the demand for higher bandwidth as compared to conventional 4K/8K video content. The 3D content is then virtually teleported and displayed to the remote participants' space [2]. In this way, content consumers can see in 6-Degree-of-Freedom (6DoF) objects (including depth) captured by the sensors cameras in a remote place within their physical space, in real-time by using AR/VR capable devices.

HTC applications have been regarded as one of the most demanding content applications for the 5G and beyond networks [3], [4]. New holographic applications are anticipated to emerge within the next few years with fully immersive AR/VR experience and near-real personal communications with holograms. Full (or near-full) immersion will be achieved when all human senses (i.e. vision, hearing, smell, taste, touch, and balance) are stimulated [5], requiring extremely high data rates (in the order of Gbps or even Tbps) to convey the rich and immersive content and even lower latency (< 20ms) for real-time user interaction [3].

On top of these requirements, high quality holographic applications require powerful devices to produce and render the frames without introducing high processing delays. To address these challenges, both academia and industry [6] are working on shoving most of the complexity from the hardware to software [7], enabling i) high-quality 3D-objects with reduced rendering latency [2] and ii) a user-aware and network-adaptive AR/VR streaming [8].

At present, the vast majority HTC applications still require consumers to wear Head-Mounted Devices (HMD) such as Microsoft Hololenses. In this scenario, the required data rate is in the order of tens of Mbps per teleported object, depending on the displayed quality of HTC objects. For example, a single sensor (e.g. Microsoft RGB-D Kinect 2.0) may require extremely high capacity i.e. $>$ 1 Gbps for each teleported object, which could be further reduced to $\approx$ 30 - 60 Mbps, by applying high-quality compression techniques [9]. The traffic generated from the sensor(s) is also directly dependent on the number of point clouds captured in the scene. In [9], the authors report an aggregated throughput that exceeds 1 Gb/s for 8 sensors, whereas in [10], the frame production server received 400 Mb/s from all 8 sensors.

Today's HTC-based teleportation applications are still in their infant status, in the sense that it is difficult to support large-scale communications over the global public Internet. This is due to the application's requirement on stringently high data rate and also the lack of agility in dealing with complex and uncertain network conditions.

In this paper, we strive to identify specific barriers and conquer them by proposing a scalable HTC based teleportation framework that can be operating at the Internet scale with assured user experiences for the very first time in the literature. We specifically target at the HTC teleportation applications requiring HMD support with the end-to-end data rate in the order of tens of Mbps per object over public Internet which has not been possible based on the existing HTC (open-source) platforms. The key novelty is that we leverage cloud-based infrastructures that are geographically close to the

ISBN 978-3-903176-28-7

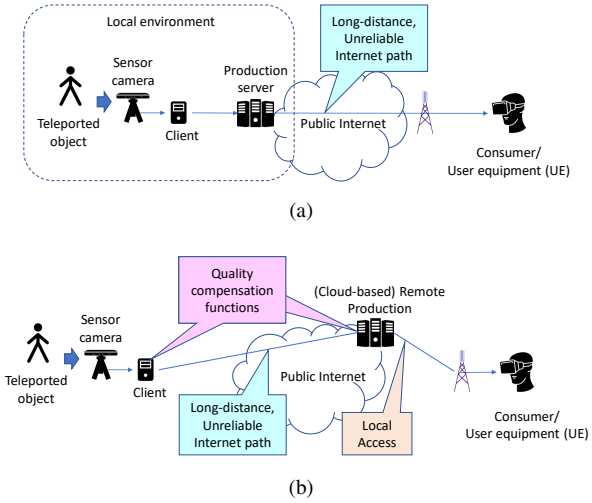

Fig. 1: High-level overview of HTC production in a) local physical machines and b) proposed cloud-based infrastructures.

actual audiences to fulfil the real-time remote HTC content production functionality rather than relying on local physical machines attached to the source to make use of the powerful cloud infrastructure and to distribute the content across the globe, as it can be seen in Figure 1. Note that in this case, the clients (i.e. the computers locally attached to the cameras for pre-processing frame data) can compress the data generated by the sensors before transmitting them to the remote cloud-based server. Hence, the bandwidth demand required for transmission over the public Internet will not be significantly higher than the traditional way with local production server. The reason to locate the cloud site for remote production function close to the audience user equipment (UE) side is as follows. Today's typical HMDs are based on wireless connections (e.g. WiFi or 4G/5G), and recent works have revealed the sub-optimal throughput performances of TCP for content delivery across hybrid long-distance Internet paths with high round trip time (RTT) combined with radio access networks (RANs) due to significantly different bandwidth-delay-product (BDP) between the two network environments [11], [12]. By locating the server function close to the audience UE side, the long RTT issue of the fixed Internet paths is circumvented. In this case, to tackle the traffic throughput uncertainty between the client side and the remote server side (which is normally all fixed network environment), we introduce sophisticated quality compensation mechanisms including frame buffering and context-aware connection management operations that can be easily supported by the high computing power in the cloud. Details on these newly proposed techniques will be introduced in Section III.

In the emerging softwarised network environments, the capability of remote HTC content production can be natively embedded in the end-to-end data delivery chain as a type of content-aware virtual network function (VNF) at the edge of the network where local audiences are attached. In order to tackle dynamic and uncertain network conditions, we propose distributed HTC frame buffering and adaptive signalling techniques that make sure the frame rate performance of teleportation streams received at the remote destination side is robust and seamless according to human being perceptions.

We carried out comprehensive real-life experiments at the Internet scale to evaluate the performance of the proposed framework and techniques. We strategically deployed 3 cloud sites across the globe, using Amazon Web Services (AWS) cloud infrastructure, at London, North Virginia and Seoul for remote HTC production, catering for local content receivers around those locations. Based on the experiments, we observed that HTC with assured QoE can be achieved by supporting buffering schemes on different sides and by applying the necessary signalling and TCP connection management techniques. By taking advantage of the cloud-based infrastructure to produce and host the content close to the audience, HTC streaming with assured QoE is also feasible at Internet-scale.

The rest of the paper is organised as follows. Section II overviews the platform applied in this work, its inherent limitations, and the challenges that need to be addressed for enabling large-scale HTC. Section III presents the development and the features incorporated in the application to support cloud-based HTC, whilst Section IV shows the end-to-end performance evaluation of the HTC application. Finally, Section V concludes the paper.

II. Traditional System and Limitations

In this work, the open-source holographic application, namely LiveScan3D toolkit [13] [1] is used as a representative HTC framework that allows to capture 3D objects and stream either the whole scene or to detect the skeletons and stream the bodies only. This software has also, a user-friendly interface, allowing the users to calibrate the system, select the type of holographic experience (i.e. whole scene or bodies only), filter out noise for reducing the effect of "flying pixels", and adjust the compression level. The latter should be used when bandwidth is the bottleneck, but would reduce the image quality.

The main advantage for LiveScan3D is the capability of the server to collect synchronised frames from various clients-sensors and to render them. That said, the server can produce a frame that displays the 3D object from different angles based on the position of the clients. Note that with the term *client*, we refer to the physical machine that the sensor is connected with. The data captured by the sensor are pre-processed at the client, which in turn sends the processed point clouds to the server upon request. The server is responsible for the 2nd level of rendering, by merging the frames received from the clients and producing the output frame. Now, that the output frame is streamed to the User-Equipments (UEs), e.g. Microsoft Hololenses.

[1]LiveScan3D: https://github.com/MarekKowalski/LiveScan3D

For each point cloud, 15 bytes are transmitted; 3 bytes for carrying the colour information (1 for each R, G, and B [2]) and 12 bytes for the coordinates (4 bytes for each dimension x,y,z). Since the application projects the colour pixels into the depth space ($512\ x\ 424$ px) and given that the frames produced by the sensor at a rate of 30 Frames Per Second (FPS), the maximum data rate required would be $(512 \cdot 424) \cdot (15 \cdot 8) \cdot 30 = 781.5$ Mbps (raw data). Of course, if data compression is applied on the client, then the data rate would be significantly reduced as mentioned in the previous section. Furthermore, since we are more interested in projecting only human beings and not the whole scene, the bandwidth required would fall below 100 Mbps for streaming one person with the current available technology. There is also some additional data transmitted (< 200 kbps for the skeleton and joints information), that could be neglected due to its small size compared to the point clouds sent.

The original system uses 11 frame types for signaling between the client(s) and the server. Six types are for control frames that are being exchanged in the beginning of the connection to correctly set-up (by exchanging the settings applied on the server) and calibrate both client(s) and server. During the calibration, management frames are exchanged for predicting the pose uncertainties in the Iterative Closest Point (ICP) algorithm as described in [13]. The calibration is initiated by the server when calibration is required (e.g. multiple sensors are applied) and it is being triggered by the user. Finally, during the connection establishment or when the user changes the settings in the server, the latter signals all the clients with the updated values (e.g. the frame carries information about the minimum and maximum bounds that will be displayed on the server, whether the whole scene or only the bodies will be displayed etc.).

The remaining 5 types are for the requests, transmissions, and signaling during the data exchanging. Note that all requests are initiated by the server, whilst the client before the actual data transmission, signals the server whether there is any data frame; a pre-recording saved frame or a live one.

Although, LiveScan3D is functional under local networks, it cannot cope with the long delays that large-scale communications over the public Internet impose, hence to support cloud-based productions around the globe. The main reasons (at the Application layer) are i) the lack of a frame buffering system to store any frames produced at both the client and the server and the UE side due to devices' capabilities, which would eliminate the possibility of losing frames between the requests and the transmission of duplicate frames and ii) the development of synchronous methods in both the Application and Network layers, introducing high delays and blocking other operations.

Both reasons would result to poor Quality of Experience (QoE) for the end users with the application stalling frequently due to the high delay introduced by the blocking operations

[2]Note that Kinect 2.0 uses RGBA, but A (which is normally for transparency) is not considered in this platform.

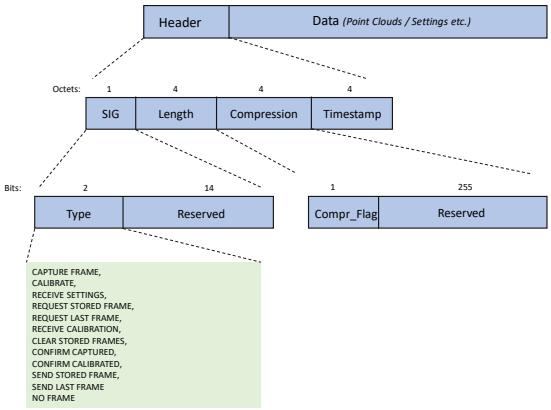

Fig. 2: Frame structure for the LiveScan3D platform, applied in this work.

and due to wasting network resources by transmitting invalid frames (e.g. replicated frames) from the Application layer. Based on the traditional approach and using the original HTC platform, we carried out complete local test with the FPS being close to 20 at the client due to the blocking operations and 25, higher than the client's, for the server due to the transmission of invalid frames. When the UE is moved to N. Virginia and Seoul (the selection of the cloud-based sites is described in Section IV), QoE severely degrades with the FPS dropping down to 7 and 3, respectively.

As for the Network layer, LiveScan3D makes use of simple and synchronous sockets based on TCP that provides sub-optimal performance as mentioned in Section I. Although, the major part of development was to introduce a frame buffering system and to improve the signaling system of the existed application, enhancements were also applied to decouple most of the functions in both the server and the client and optimise their operations in both Application and Network layers.

III. Proposed Scheme and Network Support

In this section, we present our detailed design of Internet-scale HTC framework with introduced cloud-based remote production feature with the support of necessary buffering, signalling and TCP connection management techniques. It is worth mentioning that, although we implemented our design based on the LiveScan3D platform, the proposed scheme is generically applicable to other HTC platforms. Note that in that case, the amount of data transmitted over the public Internet increases compared to the local production (due to the noise and overlapping-frame filtering occurred on the server). However, as previously indicated, thanks to the cloudification of the server production operation, the sever function can now be flexibly deployed at different Internet locations close to the audience side for local access from the server, reducing CAPEX.

The newly designed HTC framework contains data transmission, control and management operations. Control operations include the signals exchanged by the client(s) and server such as the status of the buffer and the acknowledgment of

the settings applied on both client(s) and server. On the other hand, management operations include the frames exchanged during the calibration phase, where the client(s) and server are finally tuned. Figure 2 illustrates the frame structure for the HTC platform applied in this work, with the type of a frame indicating whether the frame is for management, data, or control. The major changes from the original system are the addition of the *Timestamp* field, the introduction of additional *messages* indicating whether there are any new available frames in the client to avoid any unnecessary delays for the server waiting for frames, and the unification of all fields as a single header used in all control, management, and data frames produced by the application. To correctly set the *Timestamp*, we apply the Network Time Protocol (NTP) where both the clients and the server synchronise their clocks. This feature allows to order the frames based on the time they produced and synchronise them (in the case of multiple clients either jointly capturing a common object or separate objects).

```
Receiver Mgm (In parallel)
while frames in Rx Buffer ≤ threshold do
    Request for frames from client;
    if frame(s) exist (client) and received (server) then
        enqueue frame(s) in the Rx Buffer;
        order frames (Timestamp);
    else
        request again after X ms;
        break;
    end
end
Production Mgm (In parallel)
while display interval elapsed do
    dequeue frames from Rx Buffer(s) (Timestamp);
    if Multiple clients then
        apply frame synchronisation across all clients;
        discard any frames fallen out of synchronisation window;
    else
        proceed to the next step;
    end
    aggregate fresh incoming frames to produce the output frame;
    display of the constructed 3D object(s);
    enqueue the output frame into the Tx Buffer;
end
Transmitter Mgm (In parallel)
while request from UE(s) do
    if output frames timestamp > UE's served frames then
        select those output frames from Tx Buffer;
        transmit to UE(s);
        update Tx Buffer status;
    else
        signal UE(s) that no new output frames exist;
    end
end
```

Algorithm 1: Pseudocode of the server functionalities

There are three core functions running in parallel on the server in the cloud; *Receiver Mgm*, *Production Mgm*, and *Transmitter Mgm*, each one of them handling specific tasks and functions, as shown in Figure 3 and described with the (pseudo-algorithm) Algorithm 1. The former one is responsible for handling all information related with the connections to the clients and the frames received by them (establishment, data exchanging between the clients and the server). It is also the one that checks the buffer status and requests from the clients for new frames. The second core function is responsible for the synchronisation of the frames and the production of the output frame, which can be displayed on the server or be streamed to the UEs. In particular, the frame synchronisation is required when multiple clients / sensor cameras either jointly capture a common object or operate remotely capturing different objects. In that case, any input frames in the server will be aggregated together only if their timestamps fall into the same time window, meaning that they were originally produced at approximately the same time, otherwise the objects displayed with the output frame might not be synchronised. The latter core function, *Transmitter Mgm*, is responsible for serving the UEs and handling all the statistics with these connections, e.g. up to what frame each UE has received etc.

Furthermore, we introduce a set of distributed frame buffering mechanisms for dealing with network uncertainties especially over the long-distance Internet path. This includes a three-level buffering at the cloud-based server side as well as on the client and UE sides. Even though, the queues are based on the First In First Out (FIFO) method, the frames in all buffers are organised in ascending order based on the production timestamp. The reason of using a three-level buffering mechanism at the server is due to the three decoupled and distinguished operations taking place on the server, as mentioned earlier.

The first buffer, the *Rx Buffer* is the one that stores the frames received from a client side. The server holds one buffer per client, where based on a fixed time interval checks the status of the *Rx Buffers* and i) orders the frames based on the timestamp and ii) requests a frame from a client when the frames buffered are less than a pre-configured *threshold*. The actual system performance based on different settings of such threshold is analysed in Section IV. In particular, the server will first wait for the frames in the buffer to be at least equal to this *threshold* before starts dequeing and producing the output frame for the first time. Since we developed these novel features to enhance the receive/transmit capabilities of the devices [3] by allowing multiple parallel TCP connections per device when it is necessary for coping with high RTT [12] which has recently been proposed in other content delivery scenarios, and given that the frames received by a client regardless of the number of parallel connections to the client are enqueued in the same buffer, the frames are organised in ascending order in this first buffer. Any outdated and invalid frames are being discarded from this incoming buffer. Once the *X* ms time interval elapses, the *Production Mgm* function initiates the production of the output frames by dequeuing the frames from the *Tx Buffer* based on the timestamp and adds them to the list for rendering. Note that if multiple clients are connected to the server, then the *Production Mgm* function will only dequeue the frames belonging in the same time window based on their timestamps, hence the frames selected to be synchronised. For example, the frame with timestamp *TS1* from *Client 0* will be merged with the frame with timestamp *TS1* from *Client N*, whereas the the frame with timestamp *TS2* from *Client 0* will not be rendered with

[3]With the term devices, we refer to all devices engaged in the HTC framework (i.e. clients, server, UEs).

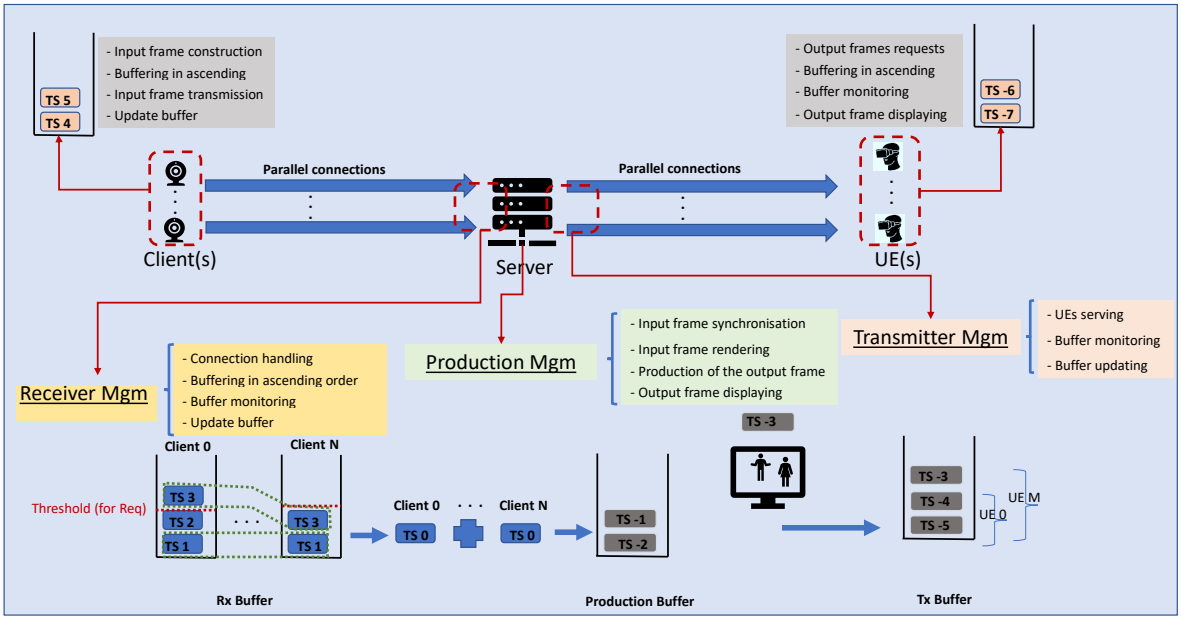

Fig. 3: HTC system development and features.

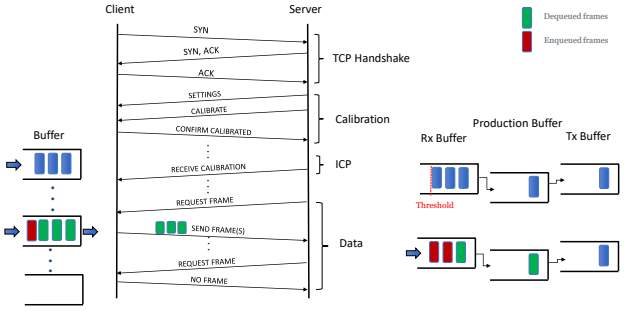

Fig. 4: Signalling exchange sequence between a client and a server.

other frames if they do not belong in the same time window (e.g. *TS2* ± 10 ms). Now, the output frame is being enqueued in the second buffer, the *Production Buffer*, and also pushed into the third buffer, the *Tx Buffer*. The server displays the frames stored in the *Production Buffer* with a rate that is similar to the one supported by the sensors (e.g. 30 FPS in our system, which corresponds to approximately every 33ms). Finally, the *Tx Buffer* holds the frames produced by the server that are ready for transmission to the UE(s). Note that to maintain low complexity, we have developed only one queue for the transmission to the UEs, where only when a frame has been sent to all available UEs is discarded by the buffer. Hence, the *Transmitter Mgm* function records the stats per UE and accordingly updates the *Tx Buffer*.

On the other hand, the main functionalities for the clients / sensor cameras include the construction of the input frames based on the object captured (1st level of rendering), the enqueuing of these frames into the local buffer, and the transmission of these input frames upon request from the server. The UE(s) monitor their buffer status and request for any output frames from the server when their local buffer is below a *threshold* in a similar manner with the *Receiver Mgm* taking place at the server.

Figure 4 depicts a signalling sequence chart of the frames being exchanged between a client and a server for the HTC system, along with the frame buffering mechanism. Once the TCP connection has been established, the client starts storing any frames produced into its local buffer. Only after the calibration has been finished, the client(s) and server are ready to exchange the data frames. The client dequeues and transmits all available frames that are stored in the buffer upon receiving the request by the server. Note that if the buffer in the client is empty, which means that all frames have been sent to the server or for any unexpected reason there have not any frames been produced, the *NO FRAME* signal is used.

IV. Real-life Testing and Performance Evaluation

Considering the complexity of such real-life performance evaluations at the Internet scale, we adopt the following experiment strategies. First of all, giving that the new bottleneck in the end-to-end content delivery path is between the client side and the remote production server side in the cloud, we first evaluate the performance at the server side across different network locations, as robust FPS performance between the client side and the server side is a fundamental prerequisite for end-to-end user QoE. The server operating on the AWS cloud is located in three strategically selected Internet locations as previously indicated in Section I, i.e. London, Seoul, and N. Virginia. Once we have verified the assured FPS performance on the server side, as the bottleneck, it can be inferred that the UEs that are geographically close to the server in the cloud

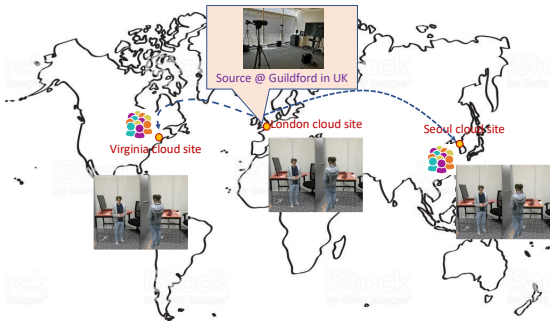

Fig. 5: Sites where the HTC remote production function is deployed.

will receive assured performance due to local content access. To verify this, we further carried out end-to-end performance based on the London cloud site where the UE is actually located in Guildford which is in south England. Note that the client/sensor [4] and the UE (for the second scenario) are hosted by the 5G Innovation Centre at University of Surrey, in Guildford, where the object is captured.

A. Remote Server Performance

Figure 5 shows a map of the server deployment across the globe and a snapshot as seen from the server at N. Virginia, where the 3D hologram is displayed. The RTT for these sites can be observed in Figure 6a, as calculated from the client. The selection of these three sites is due to the different RTT they show; very small (e.g. London with approx. 4ms), medium RTT (e.g. N. Virginia with approx. 85ms), and the extreme high RTT (e.g. Seoul with approx. 285ms).

Figure 6 depicts the performance of the HTC application in terms of FPS and throughput perceived at the remote server in the cloud for various *Threshold* values (in terms of number of frames) applied to the *Rx Buffer*; 1, 10, and 255 are the values used, with the latter one being equal to the buffer capacity used in our development. Five conclusions can be drawn from this figure. First, the client produces frames close to its maximum value, which is 30 FPS, regardless of the number of connections due to decoupling most of the functionalities used in the client as compared to the performance achieved by the original HTC framework in Section II. There is only a small variation with the site deployed in Seoul due to the high volume of transmitted traffic after a request is received at the client. Secondly, the FPS at the servers is also very close to the one observed at the client, apart from the slow start that may occur with the server at Seoul and the delay introduced with the *Threshold* value. This is also due to the decoupled functions and non-blocking operations, we developed for the server. For example, a *Threshold* of 255 frames means a delay of approx. 8 seconds (given that 30 frames are produced per second) for the displayed hologram at the server. Thirdly, this start-up delay can be reduced by applying multiple parallel TCP connections per client, especially for the case where the cloud-site operates in Seoul with the long RTT. Fourthly, 1 TCP connection is sufficient for the site deployed in London, as any additional parallel connections are hardly ever used. On the other hand, for the further sites, additional connections slightly improve the performance by increasing the throughput and reducing the start-up delay. Note that when multiple parallel TCP connections are used, the device checks the status of each connection based on a Round Robin and writes to the connections that are available. Finally, the high throughput and variation observed for the site deployed in Seoul is due to the high request interval due to the fact that the client has produced more frames that are ready for transmission (between two consecutive requests).

Figure 7 depicts the performance metrics for the buffering mechanism against the number of parallel TCP connections, the *Threshold*, and the RTT. Six conclusions can be drawn from this figure. First, by increasing the number of parallel TCP connections, the number of frames stayed inside the queues reduces due to the smoother and faster transmission of the frames over the available channels, as explained earlier. Secondly, as the *Threshold* increases, in one hand the risk of stalling reduces but on the other hand, higher delay is introduced and there is the risk of losing frames due to buffer overflow. For example, when $Threshold = 255$, the overall delay for producing and displaying the hologram due to the buffering scheme is approximately 8 - 10 sec, as illustrated in Figure 7b. In that case, the server waits until this threshold is reached before starting dequeing and producing the output frame. Thirdly, the time for a frame stayed in the queues drops with the number of parallel TCP connections. This can be seen in Figure 7c where the overall time for a frame stayed in the queues, drops of about 4000ms and 1000ms as 5 parallel connections are open against 1 and 3, respectively. Fourthly, the higher RTT significantly increases the dequeing interval from the *Client Buffer*, affecting the rest of the queues at the server and of course increasing the volume transmitted by the client after a request received, as explained earlier. Since more frames are transmitted now, there are more frames inside the server's queues that are waiting for them to be dequeued. Fifthly, this dequeing interval from the *Client Buffer* can be controlled by increasing the number of parallel connections as illustrated in Figure 7e. In particular, the maximum dequeing interval can be reduced as frames are sent faster on the available channel (the probability of a channel being available increases with the number of parallel connections, especially in links with high RTT where TCP ACK may cause additional delays and block transmissions). Finally, Figure 7f shows the processing delay introduced by the functions running on the client when only 1 TCP connection is open. Note that under the profiling mode, the client does not operate in the optimised mode, hence the processing delays will be higher than the ones during the optimised operation (i.e. profiling mode is

[4]Kinect 2.0 is the sensor and the client machine is an Intel(R) Core(TM) i7-6820HQ CPU @ 2.70GHz with 32GB.

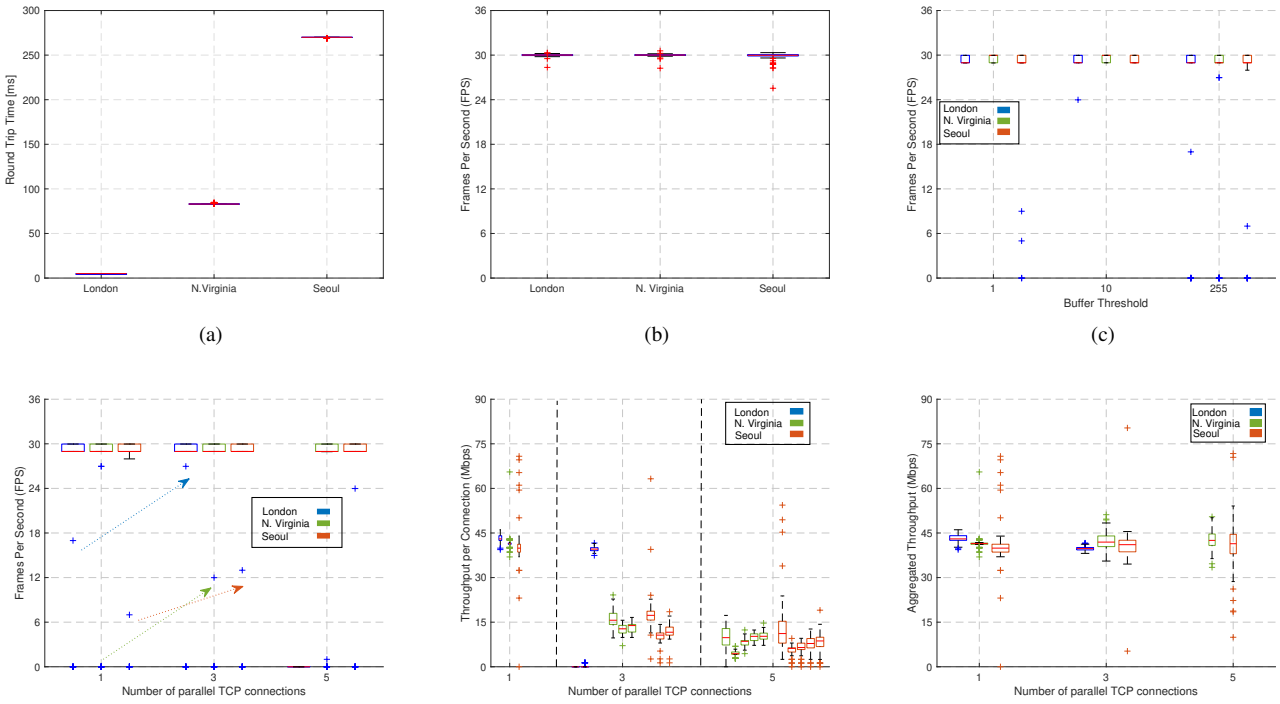

Fig. 6: Performance metrics in both client and server: a) RTT between the client and the servers, b) FPS at the client, c) FPS at the server against the frame buffering threshold (e.g. pre-fetching), d) FPS at the server against the number of parallel TCP connections per client, e) throughput perceived per TCP connections, and f) aggregated throughput perceived at the server.

disabled [5]). The list with the functions is:

- HO: responsible for getting the latest frame from the sensor and the colour, depth, and body information.
- H1: responsible for the Rx socket.
- H2: responsible for the Tx socket.
- H3: create the frames for transmission.
- H4: map depth information to space.
- H5: map colour information to space.
- H6: process the colour information and draw the data.
- H7: responsible for constructing the frame based on the depth and colour information.

Although, most of the core functions do require less than 1ms for their execution, *H0, H2,* and *H7* have the lion's share of the overall processing delay. Especially, function *H2* where huge volumes of traffic are required to be transmitted to the server. Since only 1 TCP connection is open, it is natural for this function to "wait" longer until all available frames are pushed to the channel.

B. End-to-End Performance

This subsection serves for the case where the cloud-based server distributes the content to UEs. In this scenario, the cloud-based server distributes the content (after the frame is being rendered and displayed on the server) to local consumers. In particular, the client/sensor and the UE are hosted by the University of Surrey in Guildford, as described earlier, whilst the server runs on the AWS cloud in London.

To enable the holographic experience on mobile devices, we have extended the Hololens application [6] by incorporating Google's AR technology to realise HTC on Android mobile devices. This allows the consumers to experience HTC from the screen of their mobile phones, that i) adds flexibility e.g. no need for wearing the bulky HMD and ii) supports of connectivity through cellular (remote) or Wi-Fi networks, allowing for high-scale deployments (e.g. in a stadium, concert hall etc.) pushing the limits of connectivity. On the other hand, this technology is prone to mobile phone's limitations (e.g. powerful devices are desirable). In particular, for our experiments, we use the Samsung S9 mobile device running Android 9, to support the highest applicable quality level [7] for mobile phones.

Figure 8 illustrates the performance metrics for the buffering mechanism *Server - UE*, since the performance for the *Client - Server* is not affected by the presence of UEs. Furthermore, the throughput and the FPS perceived at the UE are also presented

[5]This is another feature that we have implemented to measure the processing delay of the various functions, resulting in 4-5 times lower fps due to the logging. Of course, the performance is also affected by the capabilities of the client machine (e.g. how powerful it is). This mode can be selected by the user on the fly.

[6]https://github.com/MarekKowalski/LiveScan3D-Hololens.

[7]For the development, the Unity software was used along with the Google HelloAR framework.

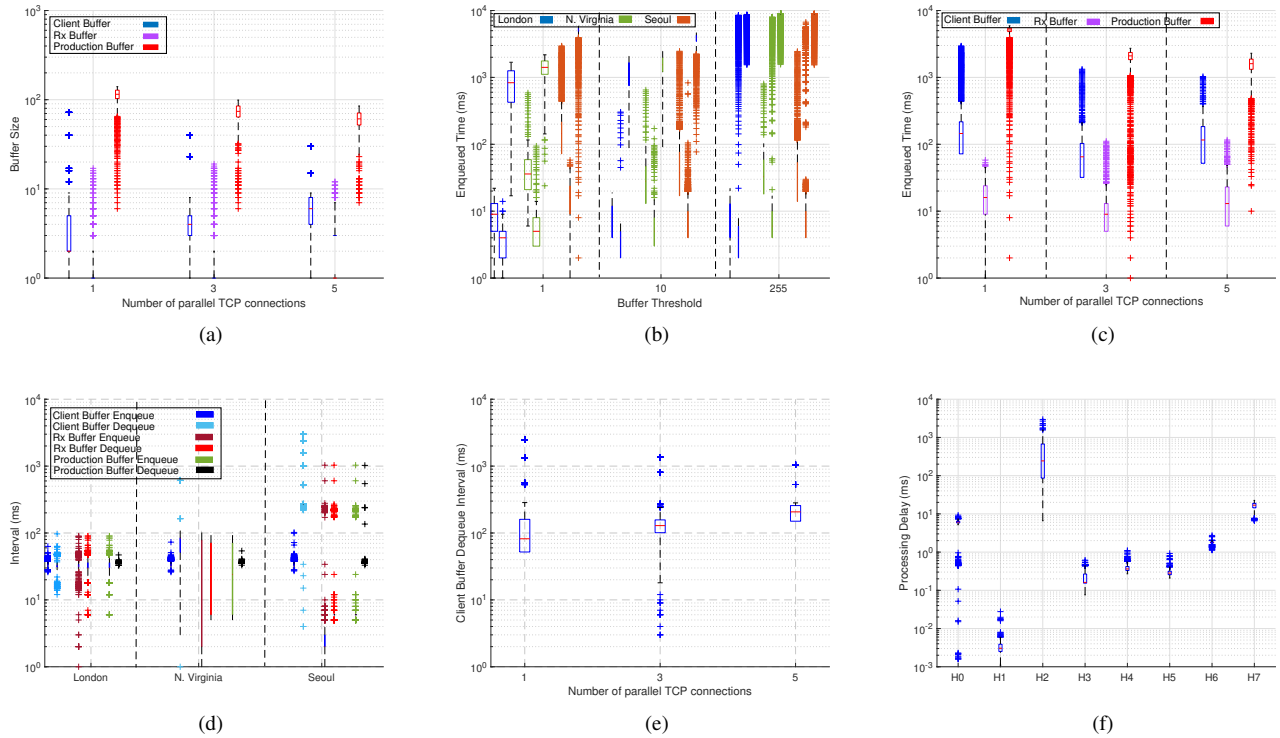

Fig. 7: Performance metrics in both client and server in respect to the frame buffering mechanism: a) number of buffered frames, b) time for a frame stayed in a queue per site and buffer (e.g. *Client, Rx,* and *Production*), c) impact of the parallel TCP connections on the time spent for a frame in the queues, d) impact of the various RTT on the enqueuing/dequeuing interval, e) impact of the parallel TCP connections on the dequeing process in the client, and f) profiling metrics for the functions running on the client.

in this figure. Three conclusions can be drawn by this figure. First, a similar behaviour is observed with the increase of the *Threshold*, as the delay inclines as well. Secondly, the higher *Threshold* implies that the UE will request more frequently for frames, resulting in higher complexity and longer delays at the *Tx Buffer*. Finally, by increasing the *Threshold* a smaller variation in terms of FPS and throughput can be observed in Figures 8c and 8d, as the *UE Buffer* has always frame for displaying and the more frequent requests. In particular, the buffering scheme, the functions' decoupling, and the use of parallel connections assure an end-to-end QoE with FPS close to 30 and bandwidth required 40 - 50 Mbps.

Even though, up to 5 parallel connections can also be supported for the Server-UE link, QoE degradation was observed by enabling more than 1 connection with the FPS dropping by approximately 20-30%. The main reason for this could be the increased complexity, that the mobile phone could not cope with.

V. Conclusions

In this paper, we have presented an end-to-end solution for enabling in internet-scale Holographic-type Communications (HTC) system, with guaranteed user Quality of Experience (QoE). The bandwidth requirement for assuring seamless and smooth streaming of the 3D teleported object is strictly depends on the object and the number of objects being captured by the camera/sensors, with the minimum being close to 40 Mbps per teleported object.

Our HTC platform, can be developed into a type of Virtual Network Function (VNF) based on Network Functions Virtualization (VNF), by leveraging cloud-based remote production and distribute the content across the globe based on where the audience is. The intelligent HTC frame buffering and signalling mechanisms developed in our HTC platform along with the context-aware TCP connection management at different segments of the end-to-end content delivery path across the public Internet (e.g. based on some context information such as the distance) are able to guarantee QoE to the end user.

By incorporating cloud-based infrastructures to remotely produce and distribute the content the network uncertainties occurred in the long-distance transmission of holographic-type content can be overcome and guaranteed Frame per Second (FPS) performance to the end user can be achieved, with the content located close to the audience. Furthermore, flexibility is improved as the HTC server can be deployed at different cloud sites and act as a Multi-access Edge Computing (MEC) server, whilst at the same time the Capital Expenditure (CAPEX) can also be reduced when compared to the local

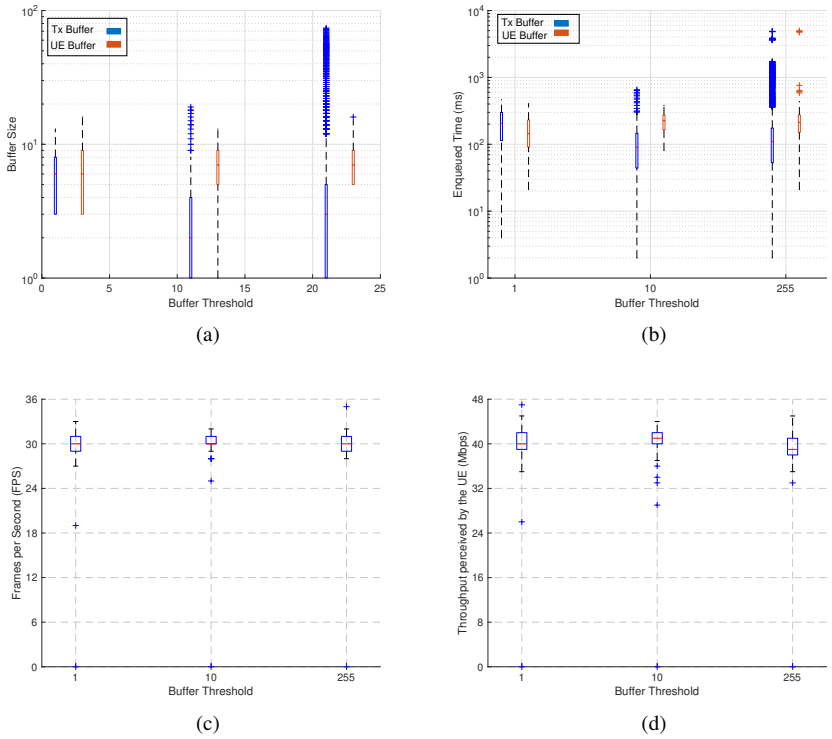

Fig. 8: Performance metrics for the UE: a) number of frames buffered in *Tx Buffer* and *UE Buffer*, b) time for a frame stayed in the *Tx Buffer* and *UE Buffer*, c) FPS as perceived by the UE, and d) throughput perceived at the UE.

production. As the immersive technology advances and the content becomes richer, networks and solutions that offer lower latency and even higher bandwidth will be required to support new immersive technologies.

ACKNOWLEDGMENT

This work was funded by Huawei Technologies, through cooperation with Network Technology Laboratory, 2012Labs. We would like to acknowledge the support of the University of Surrey 5GIC (*http://www.surrey.ac.uk/5gic*) members for this work. The authors would also like to thank the authors of the software used in this work, Marek Kowalski and Jacek Naruniec, for their helpful advice on various aspects of the software's functions. Last but not least, we would like to show our gratitude to Dr. Change Ge his valuable assistance in this work.

REFERENCES

[1] R. Li and Y. Miyake, "New Services and Capabilities for Network 2030: Description, Technical Gap and Performance Target Analysis," Doc. NET2030-O-027 in FOCUS GROUP ON TECHNOLOGIES FOR NETWORK 2030, Geneva, 2019.

[2] S. Orts-Escolano et al., "Holoportation: Virtual 3D Teleportation in Real-time," Proceedings of the 29th ACM Annual Symposium on User Interface Software and Technology, pp. 741 - 754, 2016.

[3] R. Li, "Towards a new Internet for the Year 2030 and Beyond," Third Annual ITU IMT-2020/5G Workshop and Demo Day (https://www.itu.int/en/ITU-T/Workshops-and-Seminars/201807/Documents/3_Richard%20Li.pdf), Geneva Switzerland, July 2018.

[4] D. L.-Perez, A. G.-Rodriguez, L. G.-Giordano, M. Kasslin, and K. Doppler, "IEEE 802.11be Extremely High Throughput: The Next Generation of Wi-Fi Technology Beyond 802.11ax," IEEE Communications Magazine, vol. 57, no. 9, pp. 113 - 119, 2019.

[5] S.-H Jun and J.-H Kim, "5G Will Popularize Virtual and Augmented Reality: KT's Trials for World's First 5G Olympics in Pyeongchang," Proceedings of the ACM International Conference on Electronic Commerce (ICEC'17), pp. 1 - 8, 2017.

[6] X. Li et al., "A Critical Review of Virtual and Augmented Reality (VR/AR) Applications in Construction Safety," Automation in Construction Elsevier, vol. 86, pp. 150 - 162, 2018.

[7] A. Maimone, A. Georgiou, and J. S. Kollin, "Holographic Near-eye Displays for Virtual and Augmented Reality," ACM Transactions on Graphics (TOG), vol. 36, no. 4, pp. 85:1 - 85:16, 2017.

[8] J. Park, A. P. Chou, and J.-N. Hwang, "Rate-utility Optimized Streaming of Volumetric Media for Augmented Reality," IEEE Journal on Emerging and Selected Topics in Circuits and Systems 2019, vol. 9, no. 1, pp. 149 - 162, 2019.

[9] A. D. Wilson, "Fast Lossless Depth Image Compression," Proceedings of the 2017 ACM International Conference on Interactive Surfaces and Spaces, pp. 100 - 105, 2017.

[10] A. Fender and J. Müller, "Velt: A Framework for Multi RGB-D Camera Systems," Proceedings of the 2018 ACM International Conference on Interactive Surfaces and Spaces, pp. 73 - 83, 2018.

[11] C. Ge, N. Wang, G. Foster and M. Wilson, "Toward QoE-Assured 4K Video-on-Demand Delivery Through Mobile Edge Virtualization With Adaptive Prefetching," in IEEE Transactions on Multimedia, vol. 19, no. 10, pp. 2222-2237, Oct. 2017.

[12] C. Ge, N. Wang, W.-K. Chai, and H. Hellwagner, "QoE-Assured 4K HTTP Live Streaming via Transient Segment Holding at Mobile Edge," on IEEE Journal on Selected Areas in Communications, vol. 36, no. 8, pp. 1816 - 1830, 2018.

[13] M. Kowalski, J. Naruniec and M. Daniluk, "Livescan3D: A Fast and Inexpensive 3D Data Acquisition System for Multiple Kinect v2 Sensors," Proceedings of the 2015 International Conference on 3D Vision, pp. 318 - 325, 2015.