Cloud Displays for Mobile Users in a Display Cloud

Lars Tiede, John Markus Bjørndalen, and Otto J. Anshus
Department of Computer Science
University of Tromsø, Norway
lars.tiede@uit.no, jmb@cs.uit.no, otto@cs.uit.no

ABSTRACT

The display cloud model allows users to select local and remote programmable displays, and add them to a user specific cloud display where the user can arrange them freely. On a cloud display, the abstraction representing remote graphical content is termed a visual. It can be positioned and resized freely. Wherever a visual touches a part of the cloud display with physical displays present, the physical displays will show the corresponding graphical content of the visual. The physical displays can simultaneously show several visuals from one or many users.

The display cloud approach is suitable for public environments because we do not allow user customization of the displays, a user does not have to expose any data except the actual graphical content to the display computers, and he does not have to go through the displays to do user interaction with his resources. Mobile devices have an essential role in achieving this. They provide, for each user, the means to detect displays, to add displays to the user's cloud display, to manage displays and visuals in a cloud display, and to interact with visuals.

An insight is that the display cloud model is maximally decentralized between users, and maximally centralized per user. We conducted a set of experiments on a prototype using 28 display computers with up to 21 users. The results show that the prototype reacts interactively fast for each, and scales well to many users.

Categories and Subject Descriptors

H.5.2 [**Information interfaces and presentation (e.g., HCI)**]: Miscellaneous

Keywords

ubiquitous displays, display clouds, cloud displays

1. INTRODUCTION

The research problem we focus on is how to let a user display content produced on his own computers onto one or several displays both local and remote to the user. The research challenges are to understand how to do this (i) in a scalable way with regards to performance metrics like frame rates, interactive latency, and consumed bandwidth for both a single and multiple users when the number of displays increase, (ii) in a simple way so that a user can rapidly display what he wishes, (iii) in a flexible way with regards to tiling together several displays to create a larger and higher resolution display, (iv) in a secure way demanding no or very little more trust from the user than what he has already given elsewhere, using his own computers.

The methodology applied is systems research where we research possible architectures, designs and implementations of prototype systems, and document the performance characteristics of at least one such prototype. We propose, and have partially implemented, the Display Cloud approach. A user with a mobile device can easily configure a *cloud display* composed of one or many programmable displays from a loose set of displays called a *display cloud*. Using the mobile device, the user can then securely and scalably display content produced at or controlled by his computers onto the cloud display. A user can flexibly define many content entities, termed *visuals*. When a user moves from one display to another across a room, building, city, country or continent, the visuals can follow the user or be displayed wherever the user wants as long as the displays are a part of the user's cloud display.

As a case, let's assume that Amy meets her friends at a coffee shop and wants to show them pictures she has on her home computer. There is a large display cloud enabled display above the table. Amy takes her smartphone, starts the display cloud app, and scans a unique ID, a QR code, on the display. The app then connects to the display and Amy sees it represented as a rectangle on her phone. The new display has become a part of her phone's cloud display and can be dragged and positioned relative to any other display in her cloud display using the phone. Amy then uses the phone to select a picture viewing application on her home computer as the source for a visual. The visual is represented on the phone as a rectangle, and Amy drags the rectangle onto the new display's rectangle to make it visible. The display cloud system starts a viewer on the new display, and it begins to show the content produced by Amy's home computer. Through the smartphone, she can freely reposition and resize the visual content on the large display, and interact with the home computer application. Amy's friends can now watch the pictures on the large display. After Amy is done, she removes the coffee shop display from her cloud display, and all content delivered from Amy's home computer disappears. If Amy forgets to manually delete the display from the cloud display, it will automatically happen when the smartphone discovers that Amy has moved away.

We assume that a user will always have a mobile device available. We use the mobile device to (i) determine the location of a user, (ii) detect physically nearby displays, (iii) quickly set up one

HotMobile'13, February 26–27, 2013, Jekyll Island, Georgia, USA.

or several displays for temporary use, (iv) establish connection between the mobile device, the displays and a user's remote PC so that user input is transferred to the PC without involving the display(s) and display output is transferred onto the displays directly from the PC.

We observe that private and public spaces have an increasing number of displays. The expected proliferation of Android- and iOS-based consumer televisions and displays make it realistic to expect cheap programmable displays to be ubiquitous in many environments.

The trend towards open programmable displays everywhere combined with small always present mobile devices provides for the technologies needed to let us do better than today, whether we are at home, at work, or travelling: information may be displayed everywhere, in sizes suitable for multiple viewers and a lot of information may be displayed simultaneously.

The way we perceive ubiquitous displays is different from the traditional way of perceiving displays. Traditionally, displays are output devices connected to a single computer and represent a single, closed area on which the computer places all its visual content. This one-to-one relationship between computers and displays makes it harder to use the displays in the settings we described above. We need to easily compose several physically close displays into one larger display to get higher resolution and larger size. We also need to easily move display content between displays when we move from one place to another.

There are existing approaches to use more than one display at a time, either by connecting several displays to one computer in a multi-monitor setup, or by combining several displays and computers into a configuration such as a tiled display wall[9]. With these approaches, multiple displays become accessible, and display content can be split between several displays. However, these approaches typically assume a fixed number of displays in one room, and will not support a user moving outside of the room to other displays. Limited scalability with regards to the number of viewers and the number of displays when using VNC[16]-style of centralized control of the displays is documented in [19]. While VNC can support many users viewing the same low resolution desktop, the frame rate will rapidly drop as the resolution of the desktop increases.

We can avoid the limitations and drawbacks described above and achieve better functionality and scaling if we perceive a set of displays as a continuum; an open, distributed, and decentralized display surface. We term this a cloud display. Users can define their own cloud display by composing local and remote displays from the display cloud - the set of displays with the necessary functionality to be enrolled into a cloud display. A cloud display is flexible with respect to the number of physical displays that are part of it, the number of displays that are actively displaying content, and the spatial arrangement. On a cloud display, a user can put any kind of graphical content produced at his local and remote compute resources, termed visuals. Visuals that are currently supported by our prototype are VNC desktops and images fetched from web servers. Other technologies we expect to support include video streaming and desktop sharing approaches such as Apple's AirPlay.

To achieve protection, security, and ease of use for a user when he moves between displays, we use his mobile device to incorporate nearby physical displays into the user's cloud display. A key point is that displays only have access to the graphical output, and then only through temporary capabilities so that when a user session terminates, the display(s) cannot continue to pull in data from the user's PC. Using the mobile device for user input means that we do not have to trust the displays; a display cannot easily snoop on the user's input and capture passwords or other sensitive information entered into applications.

We distinguish the Display Cloud model from research on public displays like those in [2], [3], and [14] in that we have a machine centric focus. We have not investigated how users react to different ways of doing and using public apps. Furthermore, we propose or assume no special public display apps, and we have no floor control system. We have only researched and documented (i) a system making one or several displays into shared displays that users can freely compose and use as display surfaces. It is entirely up to the users what they display and where on the displays they display it. (ii) A system where mobile devices play a crucial part to achieve security by letting the users only trust what they already trust: their own computers. (iii) A system where mobile devices provide user input to the computers running the application(s) that produce output for the displays.

We believe that the Display Cloud system can be used on public displays for advertisements and informational purposes as well as by individuals briefly needing larger displays.

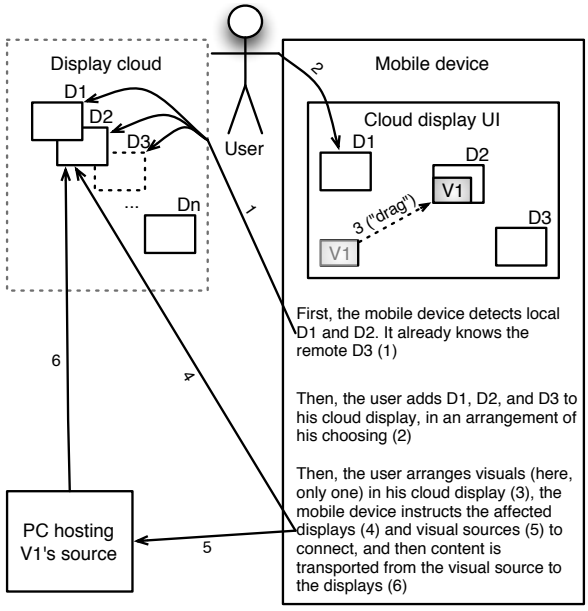

Figure 1: Display cloud, cloud displays, and visuals

2. USAGE SCENARIOS

Figure 1 illustrates the concepts used in the following usage scenarios. The scenarios hint at some functionalities and features not yet available in the prototype. However, we have found the scenarios useful in describing the range of possibilities, and they help us identify issues to be solved.

Scenario 1: In the Lab, Giving a Presentation. Ken, a visiting researcher, and his hosts meet in a lab for a presentation. The lab has a tiled display wall comprised of many displays. Ken uses his smartphone to detect and add the displays of the display wall to his cloud display, and arranges them into rows and columns in the same way as the display wall. Ken has already configured a visual using his cloud-hosted virtual machine as the visual source. On the smartphone, he now selects where on the cloud display the output should take place by moving the visual over the display wall's displays. His smartphone instructs the displays overlapping with the visual to start a viewer and display the remote content. Ken can

interactively resize and reposition the output to cover less or more area of the display wall, and the smartphone will direct the displays accordingly. After the presentation is finished, discussion starts and others move some of their own visuals to the display wall as well. Ken can resize his visual to make more display space available for the others' visuals. When the discussion is over and the participants leave the lab, their smartphones detect that they are no longer in the lab, and each display cloud app automatically detaches the displays from its respective cloud display, and their computers are told to serve no more content to the display wall.

Scenario 2: At the Hotel, Doing Remote Lecturing. Ken missed his flight back, and now has to give a lecture from the hotel room. He adds the large displays in the lecture hall as well as the hotel room display to his smartphone's cloud display. He defines three visuals: one with his laptop presentation as visual source, one with his laptop's camera as source, and one with the lecture hall's audience-facing camera as source. In the cloud display user interface, he moves the first two visuals to the lecture hall displays, and the last one to his hotel room's display. The students can now watch the lecture slides as well as a video of Ken on the auditorium's displays, and Ken can watch the students on the hotel room's display.

Scenario 3: At the Mall, Hanging Out. A group of teenagers meets at a mall. The mall has large displays everywhere. It also has sensors tracking customers. Both are made available to the mall shops and are paid for through subscriptions. For a customer, a display is complimentary to use for a brief period when standing next to it. The displays are used by the shops to display advertisements and coupon visuals, and by customers to display both their own visuals and interact with advertisement visuals.

When the mall's sensors and related analytics detect the group of teenagers, advertisement visuals looking for groups of teenage customers swarm towards displays near the teenagers and follow them around the mall. A teenager interested in an advertisement on a nearby display uses a smartphone to rapidly include the display to the smartphone's cloud display. Advertisement visuals, but not other customers' visuals, on that display automatically become available for selection. From the cloud display user interface, the teenager selects the interesting advertisement visual, and an instance of the advertisement visual is created for the teenager. The teenager can now interact with this instance of the advertisement visual. Other teenagers get their own instance, allowing each to browse and purchase products independently from each other.

After a while, the teenagers move on, their smartphones detect this and instruct the remote computers to stop sending data to the displays left behind. Alternatively, the smartphones can automatically add and delete nearby displays to their cloud displays, allowing the users private and advertisement instances of visuals to follow them around, moving from display to display around the mall. In both cases the visuals live on, and can be displayed again on other displays.

3. DISPLAY CLOUD ARCHITECTURE

To make a set of displays into a display cloud, and to make cloud displays from the display cloud, there are several primary functionalities we have discovered that we either need or should not have.

A display must be able to interact with a mobile device. The minimum functionality that must be in place for a display is to let a mobile device customize it, either by setting parameters for functionality already present, or by accepting new functionality given to it by the mobile device.

In the first case, the mobile device must trust that the display does not misbehave by, say, copying graphical content it sees to a third party. However, it cannot touch the user's original remote data or discover user passwords because these are handled exclusively between the mobile device and the user's remote computers. In the second case, the display must trust the mobile device as well. The mobile device can make it misbehave in obvious ways, like displaying unintended information from the internet, or adding the display computer to a botnet. Sandboxing can help reduce such dangers. Perhaps one day we will understand how to let the display figure out what an uploaded functionality will actually do, and based on this reject it or not.

Presently, we use the first approach, preinstalled functionality, recognizing that the mobile device cannot trust the display. This is a simple approach with easy to understand implications: the visuals can be compromised, but nothing else.

Functionality making all user data received by displays disappear from the display when the user wants to or when the user moves away from the display. The display will delete all user data it has received when the mobile device tells it to, and when the user moves beyond a certain distance from the display. Further, when a user quits using a display, the display must not be able to continue pulling in data from the user's remote computer. This is solved by letting the remote computer refuse further requests from the display, either when instructed by the mobile device or if the computer loses communication with the mobile device.

Mobile device functionality enabling it to dynamically discover nearby displays. This can be done in several ways using technologies such as visual tags, NFC, or Bluetooth. We are currently looking into using QR codes on or next to the displays to let mobile phones discover them and retrieve the URL or transport level network address of the displays.

Mobile device functionality to compose displays into a cloud display. The user tells the mobile device how the displays should be arranged. The mobile device tells each display what it should display. This is done at suitable frequencies, say, 25 times a second. Interestingly, the displays have no knowledge of each other and do not interact.

Further required mobile device functionality is to instruct the remote PC about making visuals available only to the relevant displays in the cloud display, to define clones of visuals so that the same content can be displayed on several displays, and to enable the user to interact with the remote computer through the mobile device.

A user's remote computers have functionality to provide visuals to the displays in a cloud display. Several approaches are possible: (i) Each display pulls in from the remote computer what the mobile device tells it to display. (ii) The remote computer pushes to a display what the mobile device tells it to push. (iii) The mobile device itself either pulls in visual content from the remote PC, or it tells the PC to push the content to it. The mobile device then acts as a proxy for the displays in either a push or a pull mode.

Each approach has advantages and disadvantages which we don't have the space to expand on here. Presently, we use the first approach.

4. PROTOTYPE

We have developed a functioning prototype of a display cloud. Many core functionalities are already implemented, including creating a cloud display, displaying visuals on the physical displays of the cloud display, and moving them smoothly on and between displays.

The prototype currently supports two approaches to transporting graphical content from a remote computer to the displays comprising a cloud display. In the VNC[16] approach, each display starts a

customized VNC viewer which then requests content from a VNC server running on a remote computer. We also use an approach where each display starts a simple picture viewer that requests images from a remote computer via HTTP.

Displays run a *display daemon* to make their functionality available to mobile devices. It listens for network connections from mobile devices, and starts visual viewers on behalf of them. Visual viewers only run as long as a user uses them; they are started and terminated on demand. This introduces an overhead when viewers are started, but it makes sure that no user state is left behind when the user stops using a display. It also minimizes persistent resource usage on the displays. Presently, the prototype does not discover when a user moves away, and the set of available displays is statically configured.

The *user controller* composes the cloud display, i.e., it deals with arranging displays, and creating, cloning, and placement of visuals. The user interacts with it through a user interface. At the moment, the user controller and the user interface run on a PC because we haven't yet ported them to mobile devices, but the infrastructure has the necessary support for mobile versions. Most user interaction is currently scripted to provide repeatability for experiments.

The *visual controller* manages a visual, i.e., it instructs displays to start and stop viewers, and it instructs visual viewers which region of the visual must be shown and where to display it. There is one visual controller instance for each visual that is managed by the user controller. The visual controller will be responsible for authorization features and proper interaction between users and visual sources when we add mobile devices.

The prototype is implemented in Python, except for the viewers which are implemented in C. The VNC viewer is a modified TightVNC viewer, and the image viewer is written from scratch. All components currently run on Linux.

The visual sources are unmodified TightVNC servers and HTTP servers. VNC servers support multiple viewers out of the box, so we only needed to instruct the viewers to request their separate regions to support splitting of a desktop to multiple physical displays. Our VNC clients can scale pixels to support different resolution displays. Other remote desktop and content streaming systems that we currently consider adding to our prototype will use different techniques for multi-resolution support.

5. EVALUATION

We report on a subset of the experiments we have conducted and their results. When changing a visual's position 30 times a second, we measured (i) how much time it took to move it on a single display and between displays, (ii) the consumed network bandwidth, and (iii) the CPU load on the display computers. We varied the number of users from 1 to 21. Each user had a single visual.

The computers were connected through a 1GBit switched Ethernet. To emulate an environment with many displays, we used 28 quad-core PCs with Linux, modified TightVNC viewers and 28 displays. To easily view and control the experiments, we used PCs that were located in a single room. We used the displays as if they were arranged into one row. To emulate a user's remote PC, we used a single core PC with Linux and TightVNC server 1.3.10. When increasing the number of users from 1 to 21, we used a cluster of identical PCs so that each user had their own remote PC. To emulate a mobile device, we ran a Python process on a display computer. User input was scripted to ensure repeatability: a mobile device process will 30 times a second tell a display to move a visual. The number of mobile devices was increased from one to 21, spreading them out so that a display computer would not host more than one simulated mobile device.

The results show that moving a visual takes about 8ms on a single display, and typically 150ms when a visual moves relatively slowly (below 3 m/s on our displays) from one display into another. The longer time for cross-display movement of a visual is because the mobile device tells a display to boot an instance of the visual viewer every time a visual enters a new display. When a visual moves faster than 3m/s, the time it takes to cross between displays is much longer, in some cases taking several seconds. When this happened, we observed that VNC was the bottleneck, spending most of the time doing protocol initiation. We suspect this is because we do too frequent connection establishments and teardowns.

When increasing the number of users, each with one visual, from 1 to 21 in the display cloud of 28 displays, the time to move each visual increased insignificantly.

A single visual consumed 1MB/s bandwidth, adding to 21MB/s with 21 visuals being displayed and moved. This is well below the capacity of the 1Gbit/s Ethernet we used. If physical mobile devices had been used instead of emulating them using processes, this would not have impacted the measurements significantly because the data to be visualized does not go through a mobile device, but directly from a remote PC to the displays.

Each mobile device process consumed less than 10% CPU on the display computer where it was running. Based on benchmarks we did, we estimate this to have been about 40% CPU load on a Samsung Galaxy S3, which has a quad-core ARM processor and 1GB RAM, running Android 4.0.4. Using a native application instead of a Python application is likely to be more efficient than the estimated 40%.

We have not reported on frames per second (FPS). FPS is limited by the remote graphics technology we use for the experiments, VNC. VNC frame rates have been reported elsewhere [7, 10]. Frame rates depend upon the number of pixels, the CPU of the VNC server, and available networks. In our setting we see typical VNC frame rates from 1-15 FPS while the viewers were moved and re-drawn at 30 FPS.

6. RELATED WORK

We use the term "ubiquitous display" as has been described by Molyneaux and Kortuem[13], who give an overview over possible technologies for ubiquitous displays, and research challenges.

Work on distributed displays and in particular display walls brought forth approaches to provide big centrally managed virtual displays to programmers and users, for example SAGE[8] and Distributed Multiheaded X[12]. Seamless screen sharing over several displays, taking not only different display resolutions but also geometrical distortions into account, has been investigated by Sakurai et al[17]. These systems usually have a static set of displays, whereas our approach aims at scalability of display usage and sharing to many users in many rooms, using a subset of many, possibly even remote displays.

Beyond dealing with distributed displays alone, many works have focused on enabling collaboration between users on large single or distributed displays through screen sharing and more, for example Dynamo[4], WeSpace[6], Impromptu[1], and Virtually Shared Displays[20]. These particular works differ from ours mainly in that they focus on enabling collaboration including, for some systems, file sharing, between users in one room.

All approaches above do not concern themselves with users composing their own view on the set of available displays, i.e., there is no conceptual equivalent to a cloud display. Mobile devices with their special sensing functionalities do not exist or do not play *essential* roles in their architectures. Several systems use the users' laptops as interaction device and content provider, whereas we sep-

arate these roles (although visuals can be hosted on smartphones, too). This makes our system more useful in a mobile environment because a user only needs to carry a smartphone to pull in content from anywhere. Another difference is that most of the above systems (except Dynamo) are tailored towards private environments or the workplace, not public spaces where devices and users can not be trusted. Furthermore, the referenced papers do not report on scaling with respect to many concurrent users or many displays.

A different approach to making displays available to users more dynamically is Dynamic Composable Computing[22], where logical computers are composed ad-hoc from a set of available devices. Unlike the display cloud approach, DCC goes beyond concerning itself only with display mechanics: to facilitate user collaboration, it also supports interconnecting other services such as different users' file systems and clipboards. When it comes to display mechanics, in DCC, a (stationary or mobile) device's framebuffer can be "connected" to a nearby display. However, DCC has no equivalent to a "cloud display", where users compose a virtual display landscape out of several local and possibly remote displays: while DCC includes the ability to connect several adjacent displays to form one logical display[11], this larger virtual display is managed as one large rectangular framebuffer and can therefore not be arranged as flexibly as displays in a cloud display. Further, displays in DCC are always in exclusive use by one user, whereas in a display cloud displays are always shared, and concurrent use, i.e. different users using different areas on one display, is possible.

A related idea to the former approach is to use virtualization to compose a "virtual platform" out of nearby resources with the STRATUS[5] system. Such a virtual platform can consist of a display driven by some computer, a CPU that is located on another computer (or even the user's mobile device), and other periphery in the network. The user brings a virtual machine using his mobile device, and STRATUS migrates the virtual machine onto a custom assembled virtual platform. This enables the user to not only show, but also host, graphical content, desktops etc. in the local environment, without having to rely on the user's resource-constrained mobile device, or accessing resources over long distances. However, a STRATUS virtual platform is vulnerable if any of the hosting computers are compromised; in our approach, only the shared graphical content and information about how to reach it is shared with public computers. Furthermore, STRATUS does not support swiftly moving graphical content across displays, sharing one display between several users, or using several displays to compose a cloud display.

No approach mentioned so far has its main focus on enabling users to interact with their own applications on *public* displays, which present challenges of their own. An early work in this area is the "personal server" by Want et al[21]. Here, the user's mobile device, a custom display-less prototype, hosts the user's application, makes its functionality available through a webserver, and a browser running on the public display accesses this webserver to show the application to the user. The personal server device the user carries has only very limited user input facilities, so the user normally uses the computer controlling the display for input. In consequence, the public display must be trusted with user input.

Later, when mobile device technology had progressed, Raghunath et al introduced the "Inverted Browser"[15], in which the user's mobile device pushes content to a modified web browser running on the public display. Here, the user uses his mobile device for interaction too: input on the mobile device is being forwarded to the public display. While this avoids using input devices on the public display directly, it does not alleviate the associated security threat: user input can still be compromised by the public display. Both personal server and inverted browser have in common that the user's mobile device plays the role of the application host, so that applications are confined to the resource-constrained mobile device. In the display cloud approach, however, applications can, but do not have to be hosted on the user's mobile device, allowing for more resource-demanding applications.

When cloud computing for hosting users' applications entered the scene, Satyanarayanan et al introduced Cloudlets[18], where virtual machines hosting the user's applications are synthesized near the physical location of both the user and the (single) public display. For this, the Cloudlets system uses on-site compute hardware, for example some computers in a coffee shop. A virtual machine is assembled from a "base" VM image that is available already (for example a vanilla Linux distribution) and an "overlay" image that the user brings along on his mobile device. After booting the assembled VM, the user interacts with it using his mobile device, and the public display shows the VM's display output. This approach enables the user to leverage the computation power of stationary hardware (applications need not be hosted on his mobile device), while at the same time allowing for applications that demand low latency between input device, application, and display (the application runs physically nearby). In our approach, these latencies are indeed higher, as the compute resources hosting visuals are often not co-located with the user and the public displays he is using. However, as there is no restriction in our approach to where visual sources can be hosted, a display cloud user could use cloudlets to host visual sources he then uses in his cloud display. Trade-offs to using cloudlets include that assembling a virtual machine before and tearing it down after use can take a long time, inducing a significant latency for the user. Further, a cloudlet user must trust the on-site computers with his virtual machine.

7. DISCUSSION AND CONCLUSIONS

A primary assumption of the display cloud model is that a user can only trust his own devices and not the display computers, and vice versa. This distinguishes the display cloud model from ubiquitous computing approaches where the environment detects the user and provides interaction mechanisms for him, and sometimes allows the user to upload code to customize the environment. Consequently, we believe that the display cloud model is well suited for public displays with many mobile users.

To achieve good scaling with the number of users, the display cloud model has no management and control bindings between different users' cloud displays. While visuals from different users can share physical displays and networks, no coordination between visuals from different users is done. Implications of this are that the display cloud model has no obvious limits to growth unless too many users end up using the same networks and the same displays. In the first case this can be solved by increasing the network bandwidth. In the second case we observe that it is highly unlikely that very many users will share the same displays because only a handful of users will fit physically around a display, limiting naturally how many visuals it will be asked to display. Even if very many users could use the same few displays simultaneously, they have no reason to do so because the visuals will conceal each other.

For individual users, the display cloud model relies on a strong centralization handled by a feature rich mobile device controlled and trusted by the user. The mobile device takes care of all interaction between the user and the remote computers, between the user and the displays, and frequently, say, 30 times a second, controls the interaction between remote computers and displays. Interestingly, despite all the responsibilities centralized to the mobile device, it is not a bottleneck. This is because the centralization is per user only.

Acknowledgements

Part of this work has been supported by the Norwegian Research Council, project No. 155550/420 - Display Wall with Compute Cluster, and Tromsø Forskningsstiftelse, project No. A2093 - Display Wall.

The authors would like to thank the Administrative and Technical Staff at the Department of Computer Science for valuable help.

8. REFERENCES

[1] J. T. Biehl, W. T. Baker, B. P. Bailey, D. S. Tan, K. M. Inkpen, and M. Czerwinski. Impromptu: a new interaction framework for supporting collaboration in multiple display environments and its field evaluation for co-located software development. In *Proceeding of the twenty-sixth annual SIGCHI conference on Human factors in computing systems*, CHI '08, pages 939–948, New York, NY, USA, 2008. ACM.

[2] N. Davies, M. Langheinrich, R. Jose, and A. Schmidt. Open display networks: A communications medium for the 21st century. *Computer*, 45(5):58 –64, May 2012.

[3] A. Friday, N. Davies, and C. Efstratiou. Reflections on long-term experiments with public displays. *Computer*, 45(5):34 –41, May 2012.

[4] S. Izadi, H. Brignull, T. Rodden, Y. Rogers, and M. Underwood. Dynamo: a public interactive surface supporting the cooperative sharing and exchange of media. In *Proceedings of the 16th annual ACM symposium on User interface software and technology*, UIST '03, pages 159–168, New York, NY, USA, 2003. ACM.

[5] M. Jang and K. Schwan. Stratus: Assembling virtual platforms from device clouds. In *Proceedings of the 2011 IEEE 4th International Conference on Cloud Computing*, CLOUD '11, pages 476–483, Washington, DC, USA, 2011. IEEE Computer Society.

[6] H. Jiang, D. Wigdor, C. Forlines, and C. Shen. System design for the wespace: Linking personal devices to a table-centered multi-user, multi-surface environment. In *Horizontal Interactive Human Computer Systems, 2008. TABLETOP 2008. 3rd IEEE International Workshop on*, pages 97 –104, oct. 2008.

[7] A. M. Lai and J. Nieh. On the performance of wide-area thin-client computing. *ACM Trans. Comput. Syst.*, 24:175–209, May 2006.

[8] J. Leigh, L. Renambot, A. Johnson, R. Jagodic, H. Hur, E. Hofer, and D. Lee. Scalable Adaptive Graphics middleware for visualization streaming and collaboration in ultra resolution display environments. *Ultrascale Visualization, 2008. UltraVis 2008. Workshop on*, pages 47 – 54, Nov 2008.

[9] K. Li, H. Chen, Y. Chen, D. W. Clark, P. Cook, S. Damianakis, G. Essl, A. Finkelstein, T. Funkhouser, T. Housel, A. Klein, Z. Liu, E. Praun, R. Samanta, B. Shedd, J. P. Singh, G. Tzanetakis, and J. Zheng. Building and Using A Scalable Display Wall System. *IEEE Comput. Graph. Appl.*, 20(4):29–37, 2000.

[10] Y. Liu, J. M. Bjørndalen, and O. J. Anshus. Using multi-threading and server update pushing to improve the performance of vnc for a wall-sized tiled display wall. In *Scalable Information Systems*, volume 18 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pages 306–321. Springer Berlin Heidelberg, 2009.

[11] K. Lyons, T. Pering, B. Rosario, S. Sud, and R. Want. Multi-display composition: Supporting display sharing for collocated mobile devices. *Human-Computer Interaction – INTERACT 2009*, 5726/2009:758–771, 2009.

[12] K. E. Martin, D. H. Dawes, and R. E. Faith. Distributed Multihead X design. Retrieved October 11, 2012 from: `http://dmx.sourceforge.net/dmx.html`, 2003.

[13] D. Molyneaux and G. Kortuem. Ubiquitous Displays in dynamic environments: Issues and Opportunities. *Proceedings of UbiComp*, Jan 2004.

[14] T. Ojala, V. Kostakos, H. Kukka, T. Heikkinen, T. Linden, M. Jurmu, S. Hosio, F. Kruger, and D. Zanni. Multipurpose interactive public displays in the wild: Three years later. *Computer*, 45(5):42 –49, May 2012.

[15] M. Raghunath, N. Ravi, M.-C. Rosu, and C. Narayanaswami. Inverted browser: a novel approach towards display symbiosis. In *Pervasive Computing and Communications, 2006. PerCom 2006. Fourth Annual IEEE International Conference on*, pages 6 pp. –76, march 2006.

[16] T. Richardson, Q. Stafford-Fraser, K. Wood, and A. Hopper. Virtual network computing. *Internet Computing, IEEE*, 2(1):33 –38, jan/feb 1998.

[17] S. Sakurai, Y. Itoh, Y. Kitamura, M. Nacenta, T. Yamaguchi, S. Subramanian, and F. Kishino. A Middleware for Seamless Use of Multiple Displays. In *Interactive Systems. Design, Specification, and Verification*, volume 5136 of *Lecture Notes in Computer Science*, pages 252–266. Springer Berlin / Heidelberg, 2008.

[18] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies. The case for vm-based cloudlets in mobile computing. *Pervasive Computing, IEEE*, 8(4):14 –23, oct.-dec. 2009.

[19] D. Stødle, J. M. Bjørndalen, and O. J. Anshus. De-Centralizing the VNC Model for Improved Performance on Wall-Sized, High-Resolution Tiled Displays. *Proceedings of Norsk Informatikkonferanse*, pages 53–64, 2007.

[20] G. Wallace and K. Li. Virtually shared displays and user input devices. In *2007 USENIX Annual Technical Conference on Proceedings of the USENIX Annual Technical Conference*, ATC'07, pages 31:1–31:6, Berkeley, CA, USA, 2007. USENIX Association.

[21] R. Want, T. Pering, G. Danneels, M. Kumar, M. Sundar, and J. Light. The personal server: Changing the way we think about ubiquitous computing. pages 194–209, 2002.

[22] R. Want, T. Pering, S. Sud, and B. Rosario. Dynamic composable computing. *HotMobile '08: Proceedings of the 9th workshop on Mobile computing systems and applications*, Feb 2008.