Peek: A Mobile-to-Mobile Remote Computing Protocol For Smartphones And Tablets

Uma Parthavi Moravapalle
parthavi@gatech.edu
Georgia Institute of Technology

Raghupathy Sivakumar
siva@ece.gatech.edu
Georgia Institute of Technology

***Abstract*—A mobile-to-mobile remote computing protocol for smartphones presents a user with the ability to run an application remotely and to interact with it in a responsive way, where I/O updates can be performed midstream and the results can be viewed in real time. Even though several protocols exist for desktop remote computing, we argue that these cannot be applied as-is for mobile-to-mobile remote computing. In this context, we introduce *Peek*, a remote computing protocol with i) multi-touch support, ii) context association, and iii) multi-modal frame compression. Through implementation on real devices, we show that *Peek* reduces the time taken to perform actions on a server by 62% on average, compared to Virtual Network Computing (VNC). We also test *Peek*'s multi-modal frame compression, against VNC, on datasets and show that it has the potential to reduce 30% of the bytes sent on the network.**

***Keywords*—Remote computing, multi-touch, smartphone**

I. Introduction

The adoption of smartphones (and tablets[1]) has seen an explosive growth over the last decade and in 2011 the number of smartphones shipped finally eclipsed that of the number of PCs [1]. Even the traditionally conservative enterprise sector is adopting mobile devices at a blistering pace, among which 71% are currently deploying or planning the deployment of mobile applications [2]. This adoption is driven by a clear return-on-investment in the form of higher employee productivity, reduced paper work, and increased revenue. It appears inevitable that smartphones will become the primary computing device for a majority of users in the future.

In this paper, we consider the problem of mobile-to-mobile remote computing. This involves a remote server running applications and the client having a view into the server using a remote computing protocol. Remote computing allows users to connect to a device remotely, view its screen in real time and control applications on it, while being physically away from it. With a mobile-to-mobile remote computing protocol, a user can experience a range of different application scenarios, which otherwise wouldn't have been possible. For example, a user can allow another user to access her smartphone to get help to edit an image. She can also play a game on her friend's smartphone, without being present at the same location. In addition to that, she can create virtual smartphone images on a resource rich cloud infrastructure and remotely access them to perform CPU heavy tasks. She can also help configure her grandmother's phone by controlling it remotely. The possible applications with a mobile-to-mobile remote computing protocol are hence numerous.

The use of remote desktop sharing is quite popular today and several protocols [3]–[8] exist to provide a remote view for desktops. However, smartphones are characterized by certain unique properties that make as-is application of existing protocols difficult, inefficient, and in some cases unviable: (i) Multi-touch interface: Existing protocols assume that the user interacts with the server with a keyboard and mouse. However, most smartphones use multi-touch screens, which cannot be supported by these protocols; (ii) Context association: A user interacts with her smartphone, not just through the input devices, but also the associated context through sensors (e.g., accelerometer, proximity sensor, etc.) for a rich application experience. However, traditional remote computing protocols do not associate a context to a session; (iii) Resource constraints: A high quality remote computing session requires high bandwidths and a considerable processing and memory power, which are available to desktops. However, smartphones are limited by low power processors and wireless networks (WiFi, 3G/4G) characterized by limited bandwidth.

Considering these differences, we introduce *Peek*, a mobile-to-mobile remote computing protocol for smartphones. *Peek* is application agnostic, i.e., users can remotely interact with any application on another smartphone, as if they were operating the device locally. Also, *Peek* is operating system (OS) and device independent, i.e., servers and clients can be developed for any OS and could reside in either physical devices or virtual images in a cloud (Fig 1). Important contributions of *Peek* that differentiate it from other remote computing protocols are: (i) Multi-touch support: *Peek* enables client-server interaction through multi-touch interfaces, which increases the ease of interaction. Compared to Virtual Network Computing (VNC), a popular remote desktop solution, *Peek* vastly increases the number of supported touch gestures. By implementing *Peek* on Android smartphones, we show that the time taken to remotely perform certain actions on the server is reduced by 62.8% on average; (ii) Context association: *Peek* associates sensor context to a session, which allows users to experience a wide range of applications that use sensor input; (iii) Multi-modal frame compression: *Peek* chooses a frame compression mode based on the server's CPU/memory load, rate of change of screen pixels and network bandwidth. Using synthetic datasets, we show that *Peek* can potentially reduce the bytes sent over a network by over 30% compared to VNC. To the best of our

This work was funded in part by the National Science Foundation under grants IIP-1343435 and CNS-1319455, and the Wayne J. Holman Endowed Chair.

[1]While all of our discussions apply to both smartphones and tablets, for brevity we refer only to smartphones in the rest of this paper.

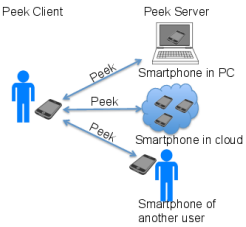


Fig. 1: Peek usage

knowledge, *Peek* is the first ever remote computing protocol designed for smartphone-to-smartphone communication. In this paper, Section I-A provides a primer on remote computing. Section II outlines the need for a mobile-to-mobile remote computing protocol for smartphones. Section III discusses the key challenges of using existing remote computing protocols for smartphones. In Section IV we present *Peek* and evaluate it in Section V. Section VI discusses related work.

A. A Primer On Remote Computing

Remote computing involves one or more client devices communicating with a server. During a remote computing session, the server encodes the content of its frame buffer (screen pixels) and sends it to the associated clients. The clients display this view on their screens and allow users to interact with it using input devices like keyboard and mouse. The clients capture these input device operations and send them over the network to the server, which executes these operations at its end and sends any screen updates back to the clients. These updates could either be a direct encoding of the screen pixels [4] or primitives such as 'draw a rectangle' [3]. The format of the messages exchanged between the client and server depends on the remote computing protocol. VNC uses Remote Frame Buffer Protocol (RFB) [4] for communication. In RFB, server encodes pixels in the frame buffer using a compression scheme negotiated between the client and server at the start of the session. The client decodes these pixels and displays them by writing onto the local frame buffer. Irrespective of the type of encoding, only those rectangles that have changed from the previous state of the frame buffer are sent over the network.

II. A Case for Mobile-Mobile Remote Computing

We consider mobile-to-mobile remote computing as a platform that extends a smartphone to a new dimension of applications. With remote computing, users can experience applications through other physical or virtual devices, and are not limited by their own device. We envision the following applications for mobile-to-mobile remote computing:

Real time collaboration: Users can collaborate on any application, even when it is not built for collaboration, by simultaneously accessing the application on a smartphone. For example, user B can help user A edit a picture on A's smartphone by remotely accessing A's device.

Computation offload: A user with a low end phone can access a virtual instance of a device hosted on a resource rich cloud and complete resource heavy tasks like panorama stitching, image manipulation, compression, encryption, etc.

Troubleshooting: A support technician can access a user's phone and help debug an issue in realtime. Such a feature would enable users with lower levels of technical expertise to use the wide range of features available with smartphones.

Client Gesture	Translation	Server execution
Tap	Left Click	Tap
Double Tap	Left Double Click	Double Tap
Long Press	Long click	Long press
Long press(short) + swipe	Left click + Move	Swipe/Drag

TABLE I: Touch to mouse translation

Multi-player gaming: Games, with or without multi player support, can be enjoyed by multiple users without being present at the same location. For example, by remotely accessing the same smartphone, two users can play Angry Birds, where they can either collaborate to pass a level, or take turns competing on the same level and compare scores.

Virtual Mobile Infrastructure: For data security purposes, certain enterprise workers are required to carry a smartphone for office use in addition to their personal phones. This can be avoided if the enterprises can provide a sandboxed virtual smartphone environment on a cloud, which the employees are allowed to access only at the workplace or with a secure VPN.

III. Key Challenges

Desktop remote computing protocols, when applied as-is for smartphone-to-smartphone remote computing present with some problems. In this section, we outline these challenges[2].

A. Input Handling

With the advent of smartphones, interactions through keyboard and mouse are no longer relevant as majority of smartphones are equipped with multi-touch screens. However, there are no remote computing protocols specifically designed for touch screen input devices. Existing VNC smartphone client and server applications are designed assuming the other end of communication is a desktop that can be controlled through keyboard and mouse operations. They adapt to the desktop protocol by translating touch screen operations into mouse operations, rather than supporting them natively. A VNC smartphone client that translates touch to mouse operations can be used to communicate with a VNC smartphone server that converts the mouse operations back to touch operations. We identified this translation between touch and mouse operations and present it in Table I. However, such a translation creates the following problems:

(i) *Many operations cannot be mapped*. Multi-touch enables a smartphone user to interact with her device using intuitive gestures performed with multiple points of contact. Since a mouse has only one pointer moving across the screen, many multi-touch operations cannot be mapped.

(ii) *Mapped gestures are non-intuitive*. For example, when a user wants to scroll a list on his smartphone, he simply swipes upwards on the touch screen. However, if the user wants to do the same on the VNC server smartphone, he has to long press the screen first and then swipe. This usage is difficult to remember as it is not natural and hence is a source of confusion. Also, if the user doesn't swipe on time after she long presses, a menu might open up corresponding to the long press gesture. Gestures that are either non-intuitive or cannot be mapped are shown in Table II.

(iii) *Context information is not associated.* A typical

[2]While we use VNC as a representative desktop remote computing protocol, the discussion can still be applied to all the other protocols.

Swipe	Scroll	Multi-finger Swipe
Multi-finger hold	Multi-finger drag	Pinch
Multi-finger tap	Multi-finger pinch	Expand
Multi-finger expand	Multi-finger multi-tap	Fling
Multi-finger Fling	Multi-finger rotate	Anchoring

TABLE II: Non intuitive and non existent gestures

Type	fps	CPU (%)	Memory (MB)	Bytes per frame
Tight 256 Color	22	48	28	13946
ZRLE	5	21	17	42101

TABLE III: VNC compression on smartphones

smartphone is equipped with many sensors including gyroscope, accelerometer, magnetometer, proximity sensor etc., which provide contextual information on the device. This sensor context is used by many applications to provide a rich experience to users. For example, there are many racing games that make use of the gyroscope and accelerometer readings to emulate effects of steering. We argue that, *apart from input devices like touch screen, mouse etc., sensors are also a way to interact with the device.* Current remote computing protocols only consider input devices like mouse or keyboard as a way of interacting with the device. Applications requiring client's sensors will not be able to operate in this scenario.

Challenge 1: How does a remote computing smartphone client interact with the smartphone server using multi-touch operations and associate its sensor context to a session?

B. Remote View Sharing

Usage of frame compression techniques designed for desktops, for smartphone-to-smartphone remote computing, results in poor resource utilization in smartphones. Some problems with using VNC for smartphones are as follows:
(i) *Frame compression in VNC is independent of device status.* Compression schemes that have a low CPU overhead have poor visual performance and those that have better visual performance are CPU heavy. To demonstrate this, we setup a VNC server that generates frames at 60 frames per second (fps) and a client that views and measures rate of display of these frames, on two LG Nexus 5 smartphones. Table III shows the fps at client and CPU, memory, bytes per frame sent at the server for two popular VNC compression schemes. 'Tight' achieves better fps (22 vs 5) and better per-frame compression than 'ZRLE', at the cost of higher CPU and memory usage. Using the same frame compression scheme throughout the session irrespective of the device status is not suitable for smartphones, as resources are limited. For example, if the server's CPU utilization is high and network utilization is low, data could be sent over the network with a simple compression scheme like ZRLE, even though a more complex scheme was fixed beforehand.
(ii) *Applications have different compression requirements.* Rate of change of screen content (frame rate) varies among different applications available. Fig. 2 shows the average size difference between consecutive frames, in a single session for different applications on Android. Frame rate for graphically intensive games like Temple Run and Minion Rush, is high compared to other applications like BBC, Ebay and Amazon. VNC uses the same type of compression scheme, irrespective of the application. Using a complex compression algorithm, which was decided at the start of session for applications with a lower frame rate, leads to wastage of resources.
(iii) *Application usage behavior leads to inefficient resource utilization in smartphones.* Many smartphone applications use a fixed number of screen layouts with different content. For example, in BBC (Fig. 3) ,articles share the same layout, which is different from that of the home screen. Consider a case in which user opens article A from the homescreen H, switches back to homescreen and opens another article B. Since a VNC server transmits the changes from the last displayed frame, the size of updates sent are U = diff(H,A) + diff(A,H) + diff(H,B), where diff(x,y) is the size of the screen update if the screen changes from x to y. If the client and server can remember those frames that could repeat (like H), or one representative frame that is similar to many frames that could be displayed in the future (say A or B), the number of bytes sent over the network could be significantly reduced (update size = diff(H,A) + diff(A,B)<<U).

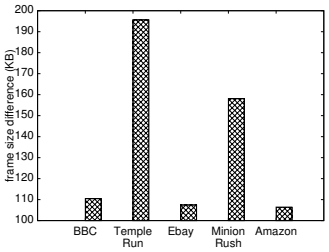


Fig. 2: Average consecutive frame size difference

(a) Article A **(b) Article B** **(c) Home Page H**

Fig. 3: Different screen layouts of BBC

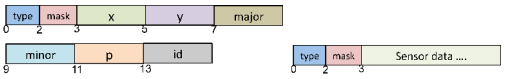
Challenge 2: How can the remote computing server compress its screen, taking into account (i) CPU, memory and network loads, (ii) screen redundancy and (iii) varying frame rate?

IV. PEEK: A MOBILE-TO-MOBILE REMOTE COMPUTING PROTOCOL

In this section, we present *Peek*, a mobile-to-mobile remote computing protocol that is (i) OS independent, (ii) application-agnostic, (iii) device independent. *Peek* is built on the RFB protocol and adds multi-touch support and context association to it. *Peek* also improves upon frame compression of RFB, by using a multi-modal compression scheme. *Peek* deals with the challenges described in Section III as follows:

A. Multi-touch Support and Context Association:

If a user has to remotely access a smartphone using VNC, she has to use a client application on her smartphone that maps touch operations to mouse operations, and a server application on the remote smartphone that translates mouse operations to touch operations. The mapping between touch and mouse operations affects the usability of the application. *Peek* clients, instead, directly capture touch interactions, represent them in a suitable format to avoid loss of integrity, and send them to the *Peek* server for execution. By removing the layer of mouse translation, touch interactions can be natively represented at



(a) Touch message **(b) Sensor message**

Fig. 4: Message format in *Peek*

the client and easily interpreted at the server. This enables the users to interact with the remote server intuitively in the same way as they would interact locally with their smartphones.

Peek adds a new touch screen input method to the RFB protocol. *Peek* clients represent each point of contact of user's finger to the screen with a touch message (Fig. 4a). Each touch message is 14 bytes long. The first byte, $type$, is a constant (=12) for all touch messages, irrespective of the device. It serves as an indication to the server to interpret the next 13 bytes of the stream as a touch message. The second byte $mask$ is a bitwise mask that represents the validity of different fields in the rest of the message. In *Peek*, touch contacts are assumed to be elliptical in shape[3] and each touch contact is represented by: (i) position on the screen - x,y (horizontal and vertical coordinates of the center of contact); (ii) dimensions - $major$, $minor$ (lengths of major and minor axes); (iii) pressure of contact - p; (iv) Id of the point of contact - id; While parameters x, y, $major$, $minor$ and p are designed to represent the physical aspects of contact, id is useful in a multi-touch scenario to differentiate one point of contact from the other. These parameters are captured in real time by *Peek* clients. The presence of x, y, $major$, $minor$, p and id in the touch message is indicated by setting bits 1 to 6 of $mask$, respectively. While simple actions like tap have only one point of contact, other actions (swipe, drag, etc.) have multiple points of contact along the path a finger traces on the screen. Each such touch contact is packed into a touch message. A special message with a mask of '0' is sent to signal the end of an action and is generated when the point of contact leaves the touch screen. When there are multiple points of contact for a touch gesture, some of the parameters might remain the same for these contacts (e.g. p). These parameters can be skipped in subsequent messages and the mask is set appropriately. The *Peek* server extracts touch parameters from the message and virtually applies the touch contact. If the $mask$ indicates that a parameter is not present, the last known value is used.

Peek clients also capture various sensor readings and send them to the server. A user with *Peek* client has an option to choose either her own device context or the server's context during a session. Fig. 4b shows a generic sensor message format. $type$ value varies with the function of the sensor (15 for gyroscope, 16 for accelerometer, 17 for proximity sensor, etc.). Similar to $mask$ in a touch message, $mask$ in the sensor message is a bitwise mask that represents validity of the sensor data. The exact format of representation of sensor readings in a sensor message depends on the type of the sensor. For example, for a gyroscope (accelerometer), it is a series of three double values, representing the rate of device's rotation (acceleration) along x,y and z axes. For a proximity sensor, it is a binary value, representing if the phone is near/away.

All the major smartphone operating systems provide APIs to interpret touch/sensor activity (e.g. UIApplication class in iOS, and /dev/input/event virtual file system in Android). Also, the implementation of extraction and execution of touch/sensor messages depends on the OS of the device. For a linux based OS, this can be achieved by writing a series of bytes into /dev/input/event virtual filesystem in a suitable format. Since message format is independent of the OS, clients and servers on different OS can communicate with each other. With this message representation, all possible multi-touch and sensor events can now be captured and communicated, thereby increasing the ease of interaction for users.

[3]Most of the touch sensor drivers assume the area of contact is an ellipse.

B. Multi-modal Compression:

Peek introduces a new multi-modal frame compression technique that takes into account content redundancy, rate of change of application content and the device resource usage. *Peek* server identifies certain key frames from the past session history and compresses the difference between current frame and a key frame that is closest to the current frame. In this way, *Peek* reduces the amount of data to be compressed, thereby reducing the amount of data sent over the network and CPU cycles. Also, *Peek* uses video compression techniques when it detects rapidly changing screen content.

While a VNC server uses a compression scheme selected at beginning of the session, *Peek* server selects one of three compression modes by periodically monitoring its CPU/memory load, network load and frame generation rate.
(a) ***last_diff:*** Like in RFB, the difference between the last frame and the current frame is compressed.
(b) ***key_diff:*** To save bytes sent on the network in a scenario where the current frame could be very similar to content in the past, the *Peek* server identifies some representative (key) frames in the session history. If the current frame is similar to any one of these key frames, the difference between the key frame and the current frame is compressed and sent along with the index of the key frame. *Peek* uses clustering techniques to identify these key frames. Frames from the session history between a particular server and client, that are similar to each other are clustered into groups using fast online integer K-means clustering algorithm[4]. For each cluster, a frame with the lowest possible difference with the centroid of that cluster is considered as a cluster head. The number of clusters to be formed is chosen based on the current memory utilization of the server and client. Periodically, cluster heads are communicated to the client and are stored as key frames in the memory of both the client and server. This overhead is negligible because cluster heads only need to be communicated infrequently. Without changing the compression algorithm, *key_diff* reduces the burden on device's resources by reducing the amount of data to be compressed.
(c) ***video_diff:*** In this scheme, the session is treated as a motion video and MPEG4 compression is used on it. This compression scheme is particularly designed for graphically intensive applications like games, which have a rapidly changing frames. For these applications, a user is presented with new content that is quickly generated through dedicated GPUs. Using *key_diff* that relies on session history doesn't make sense for these applications as the content is non repetitive. *Peek* utilizes motion prediction and motion compensation algorithms provided in the MPEG4 standard to compress these frames.

[4]While *Peek* uses K-means, it is one among a broad set of fast and light online clustering algorithms that could be used potentially

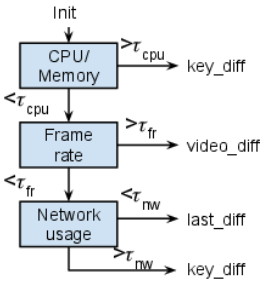


Fig. 5: Multi-modal compression

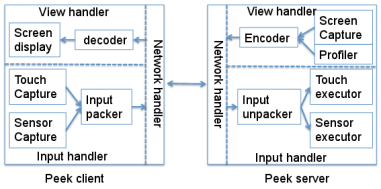


Fig. 6: System architecture of *Peek*

Action	Description
A1	Crop a picture using Photo Editor Pro
A2	Write 'A', 'B' and 'C' with finger
A3	Play 6 moves in Candy Crush
A4	Find an email in a list and delete it
A5	Open Youtube and search for 'apple'
A6	Select a paragraph in an email
A7	Find phone's IMEI number from settings
A8	Open a document and append text to the end
A9	Draw a 3x3 grid on screen
A10	Draw a smiley face on the screen

Fig. 7: Action descriptions

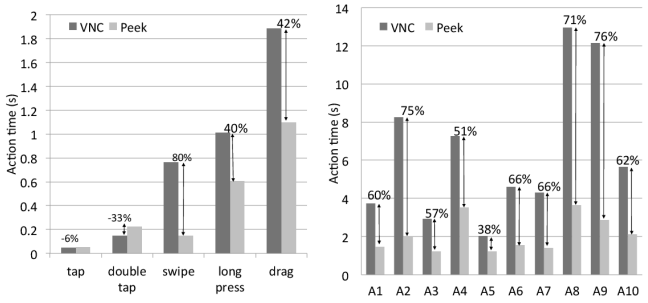
Peek server continuously monitors the device and chooses one among the three compression modes based on Fig. 5. Since CPU/memory is the most important resource that affects a device's usability, not just for remote computing, but for all other applications, *Peek* first considers the CPU/memory utilization to select a mode, and chooses *key_diff* if it is beyond a threshold τ_{cpu}. Otherwise, if frame rate is greater than a threshold τ_{fr}, *video_diff* is used. If frame rate is less than τ_{fr}, *last_diff* or *key_diff* is chosen depending on whether network utilization is less or greater than a threshold τ_{nw}.

C. System Architecture

Devices running *Peek* have three components: (i) Input handler, (ii) View handler and (iii) Network handler. The functions of these handlers change depending on whether the device is running in the server mode or the client mode (Fig. 6). The client input handler captures all the touch events and sensor events through the touch capture and sensor capture module, respectively. These modules pass the event information to input packer, which packages it into messages. The input unpacker module in server input handler unpacks the messages, interprets the parameters and sends touch/sensor parameters to Touch/Sensor executor which executes them. Server view handler uses the frame capture module to capture the device's screen. Frame encoder choses the right compression technique for a particular frame based on inputs from the profiler on CPU/memory, network and frame rate, and compresses the frame. At the client view handler, the frame decoder decodes the frames and the frame display displays it on the screen. The client/server network handler is responsible for communication between server and client over the network.

V. Evaluation

We implemented and evaluated *Peek* server and client on two LG Nexus 5 smartphones with Android v4.4.4. We build *Peek* on Android VNC Viewer [9], an open source VNC client and DroidVNCServer [10], an open source VNC server to handle touch and sensor messages. The two smartphones are connected to the same WiFi AP. We evaluate the usability of *Peek* by performing certain actions through the client on the server, and measuring the time taken by the client to generate touch messages to be sent to the server for these actions. We obtain this time by collecting network packet traces at client, filtering them for all touch/mouse message packets with server as the destination and measuring the time difference between the first and last touch/mouse packet. We also compare *Peek* with VNC by installing a unmodified VNC client and server on LG Nexus 5 smartphone and Samsung Galaxy tablet, respectively. Here, we use a tablet instead of a smartphone



(a) Primitive Actions **(b) Complex Actions**

Fig. 8: Action times

as the unmodified server application is incompatible with smartphones. Note that the method to measure time taken for an action at the client through network level traces eliminates for any bias related to the network conditions and processing power of the server. We also evaluate *Peek* only on tasks that can be performed on a tablet and a smartphone in the same way, to avoid any bias related to screensize. Therefore, we believe that the server device configuration has no bearing on the action times. Also, to discount for any user bias, we consider the average of 10 measurements for each action.

To benchmark *Peek*, we first consider a primitive action set: tap, double tap, swipe, long press, and drag. We can observe from Fig. 8a that *Peek* reduces action times significantly for certain actions. For swipe, long press and drag, the reduction is 80.2%, 39.9% and 41.8%, respectively. According to [11], an action time increase $>$ 150ms results in a noticeable reduction usability. For tap and double tap, action times are higher for *Peek* by 3ms and 74ms, respectively. This is because unlike the other actions, these actions are mapped as-is, even without multi-touch support. However, this difference does not affect the usability. To evaluate benefits of *Peek* during regular smartphone usage, we consider a set of complex actions (Fig. 7) that spans common touch screen usage patterns. For these actions, *Peek* reduces the action time by 62.8%, on an average (Fig. 8b). *Peek* achieves this by eliminating mouse mapping and directly capturing and executing touch actions. We also measured the CPU and memory usage of *Peek* and VNC on the client and observe that *Peek* does not involve any additional overheads. For a proof of concept, we also implement proximity sensor context association and verify its function. In the interest of brevity, we do not present these results here.

Next, we demonstrate the potential of multi-modal compression of *Peek* to reduce the bytes sent over the network. We consider the following applications: (a) Ebay (commerce), (b) Google play (commerce), (c) BBC (news), (d) Gmail (pro-

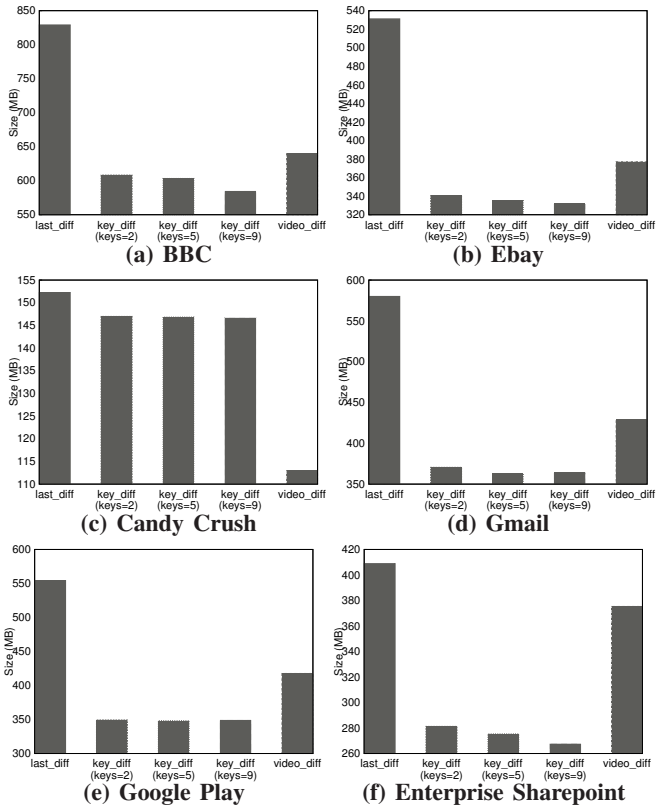


Fig. 9: *Peek* multi-modal compression

ductivity), (e) Candy Crush (games), (f) Enterprise Sharepoint (enterprise). This set is a representative mix that spans some popular application categories. We collect large usage videos for these applications and extract all distinct video frames. Test remote computing sessions of size 5500 frames are generated from these distinct frames as follows - each frame is either chosen randomly from the set of distinct images or is the same as the previous image, with equal probability. This dataset represents typical usage behavior where in a user either sticks with the current view or interacts with it (with 0.5 probability) and hence provides a way to evaluate *Peek* during random user behavior. Since the collection of large real application user traces for many applications is highly intrusive, we use synthetic datasets for evaluation. This is because recording screens and writing them to the storage card, while the user is using an application involves a lot of I/O operations and is a CPU heavy task. We implement and evaluate different modes of multi-modal compression used in *Peek* on this synthetic dataset in Matlab. For *last_diff* and *key_diff* we use Tight PNG, a popular compression used in VNC, to compress the difference between two frames.

Fig. 9 shows the post-compression dataset sizes after using different modes of *Peek*'s multi-modal compression. We can observe that using *key_diff* and *video_diff* results in better compression. For example, compared to *last_diff* (used in VNC), *key_diff* with 5 key frames results in a reduction of dataset size by 27.2%, 36.8%, 37.4%, 31.4% and 32.6% for BBC, Ebay, Gmail, Google play and Enterprise Sharepoint, respectively. For all applications, *last_diff* results in highest post-compression sizes. For Candy Crush, *video_diff* performs the best. This is because it is a game having rapidly changing screen content, with little repetition from the past as the user advances to new levels. We can also observe that, increasing the number of stored key frames results in better compression. However, this reduction is not prominent and considerable benefits can be achieved by using just 2 key frames.

VI. Related Work

There are several remote computing protocols for desktops in use today. For example, RDP (Microsoft) [3], RFB (VNC) [4], RGS (HP) [5], ALP (Sun/Oracle) [6], ICA (Citrix) [7], PcoIP (Teradici) [8], etc. For mobile thin clients, some optimizations have been proposed in related literature. SmartVNC [12] reduces the burden of doing tasks in a remote computing session from smartphone to a desktop, by identifying macros. Mobidesk [13] proposes WAN traffic optimization for mobile thin clients. Modeap [14] uses translation between graphical primitives of desktop and those of a mobile web browser. [15] and [16] are other solutions that target gaming and multimedia delivery on smartphones, respectively. However, all these solutions assume the server is a desktop.

VII. Conclusion

In this work, we presented *Peek*, a remote computing protocol for smartphone to smartphone remote computing with multi-touch support, context association and multi-modal frame compression. We evaluated *Peek* and show that it reduces the time taken to perform certain actions by over 60% and has a potential of reducing the number of bytes transmitted into the network by over 30%, compared to traditional VNC. To the best of our knowledge, *Peek* is the first ever mobile to mobile remote computing protocol for smartphones.

References

[1] "Smartphone sales exceed those of pcs for first time, apple smashes record," http://www.digitaltrends.com/mobile/smartphone-sales-exceed-those-of-pcs-for-first-time-apple-smashes-record/.

[2] "Symantec survey," www.symantec.com/about/news/release/article.jsp?prid=20120202., Feb 2012.

[3] "Remote Desktop Protocol," http://msdn.microsoft.com/en-us/library/aa383015(v=vs.85).aspx.

[4] "Remote Frame Buffer Protocol," http://www.realvnc.com/docs/rfbproto.pdf.

[5] "Hp remote graphics software," http://www8.hp.com/us/en/campaigns/workstations/remote-graphics-software.html.

[6] "Appliance link protocol," https://docs.oracle.com/cd/E35310_01/E25747/html/security-alp.html.

[7] "Citrix reciever, http://www.citrix.com/go/receiver.html."

[8] "Pcoip technology," http://www.teradici.com/pcoip-technology.

[9] "android-vnc-viewer, vnc viewer/client for android platform," https://code.google.com/p/android-vnc-viewer/.

[10] "Droid vnc server," https://github.com/oNaiPs/droid-VNC-server.

[11] N. Tolia *et al.*, "Quantifying interactive user experience on thin clients," *Computer*, vol. 39, no. 3, pp. 46–52, March 2006.

[12] C.-L. Tsao *et al.*, "Smartvnc: An effective remote computing solution for smartphones," in *Proc. 17th ACM MobiCom*, Las Vegas, NV, USA, 2011.

[13] R. A. Baratto, S. Potter *et al.*, "Mobidesk: Mobile virtual desktop computing," in *Proc. 10th ACM MobiCom*, New York, NY, USA, 2004.

[14] H. Li *et al.*, "Modeap: Moving desktop application to mobile cloud service," *Mobile Networks & Applications*, vol. 19, no. 4, pp. 563–571.

[15] C.-Y. Huang, K.-T. Chen *et al.*, "Gaminganywhere: The first open source cloud gaming system," *ACM Trans. Multimedia Comput. Commun. Appl.*, vol. 10, no. 1s, pp. 10:1–10:25, 2014.

[16] B. Joveski *et al.*, "Semantic multimedia remote display for mobile thin clients," *Multimedia Systems*, vol. 19, no. 5, pp. 455–474, 2013.