Multi-Player Gaming in Public Transport Crowd: Opportunities and Challenges

Saumay Pushp†, Chi Harold Liu‡, Fangming Liu† and Junehwa Song†
†KAIST, Daejeon, Republic of Korea, ‡Beijing Institute of Technology, China
†‡Huazhong University of Science and Technology, China
E-mail: †{saumay,junesong}@nclab.kaist.ac.kr, ‡chiliu@bit.edu.cn, ‡fmliu@mail.hust.edu.cn

Abstract—Smart devices are supporting emerging types of location-based gaming applications to attain collaborations among collocated users in public places like transports. However, they are facing many challenges like the timely performance of back-end game servers for runtime game operation, unreliable cellular network connections, and opportunistic and dynamic local environment. To address these limitations, in this paper, we propose “CrowdMoG”, a Crowd based Mobile Gaming platform, to identify and match the nearby passengers on the move according to their associated gaming preferences, and provides the smooth session handoff to enable the continuity of existing game plays when participants leave the game due to different mobility patterns. We therefore describe the potential, along with challenges and opportunities that open up the new dimension for the entire research community to redesign and examine a tradition problem, fundamentally transforming it into a new era of mobile gaming experience.

I. INTRODUCTION

In many crowded metropolitan cities, people often travel by public transports like metro and local buses everyday. A finding shows that about 3,000 commuters in the west of England spend about 139 hours a year on average traveling to and from their work place [1]. Therefore, mobile computing in crowded public transports can open up a new opportunity to harness a collection of smart devices (e.g., smartphones and tablets [2], [3]) in their vicinity as a unified computing substrate. Figure 1 shows a snapshot of the Beijing Underground where most of crowded passengers are looking at their smart devices. A promising aspect we focus in this paper and our research path in its own right, is to enable highly-interactive and opportunistic multi-player gaming [4]–[6] among the crowd. Examples of these games are first-person-shooting (FPS) and racing. In it, users play and exchange their game play information dynamically among the crowd, however they come with many challenges.

First, despite the fact that mobile gaming has been a dominant entertainment source while they commute in public transports⁠, and the increasing number of mobile games are available, their adaptability as multi-player games is still in an early stage. As far as we are concerned, none of them are carefully designed for the highly-crowded public transport environments, as they are either single-player based or turn wise multi-player, and these applications are usually built with suboptimal features, like for a particular platform or network type.

Second, in public transports, cellular network connectivity like 3G UMTS [7] and 4G LTE [8] is not reliable, with high link variability in terms of latency, jitter, and bandwidth. Thus, it is extremely challenging to provide high quality gaming experience to the end user, without intermittent, abrupt response and interruptions. Therefore, traditional client-server based gaming model via cellular networks is not feasible that may hinder the game join and runtime smooth operation. If using broadcast, given that many gaming groups may potentially exist in a crowd, broadcasting from each end terminal would cause heavy co-channel inter-group interferences. This problem is more severe for certain type of game genres like FPS and racing, requiring frequent and time-critical interactions among players and thus their tight latency requirements have to be strictly guaranteed.

Third, traditional cloud-based gaming services use game engines [9], [10], where mobile clients are only responsible for sending game-related commands to the cloud server, while the latter coordinates nearly all aspects of game operations. Although this centralized approach works on PCs, it suffers in our scenario where a stable Internet connectivity cannot be assumed. Furthermore, the concept of realizing a gaming cloud Platform as a Service (PaaS) restricts the on-the-fly gaming behaviors of individual passengers, where the availability of the service is a must prerequisite.

Lastly, the multi-player game play is further complicated by the mobility pattern of both moving vehicles and passengers’ different routine information, e.g., the duration between two stops, the time window on the platform, where to board/get off the vehicle, etc. Thus, the question still remains on how to estimate the stability of an existing gaming group so that anyone’s departure will not impact on the overall gaming experience of others, and the newly arrived participants can accurately select the right game group and competitors among...
the crowd.

Finally, traditional ways like constant scanning and retrieving information from the cloud server in finding participants of the proximity may cause additional signaling overhead and energy efficiency issues. Furthermore, the conventional lengthy registration/sign-on/involvement process over the back-end game server is obviously not applicable either. Therefore, a distributed game operation within the local crowd is strongly desired.

To address the above limitations and enable highly interactive gaming in crowded public transports, we have been developing "CrowdMoG", the Crowd based Mobile Gaming platform. It should not only provide users with a distributed open source game execution engine, but also reduce the network latency by efficient protocols. As a core functionality, CrowdMoG aims to provide a location-aware matchmaking process (supported by multiple modules) to identify and match a passenger with his/her preferred game group. It goes around the hassles from the traditional client-server based gaming model by using cellular network connections, but continuously overhears the network packets [11] during game operation from the game owner (GO), as the coordinator of a game play group. It offers low signaling overhead without requiring each participant to constantly broadcast the game information in the crowd. Nevertheless, traditional network overhearing does not consider how to guarantee the game session continuity (and handoff, e.g., switching the GO upon his/her departure) imposed by passenger dynamics, under the stringent time constraints.

To summarize, the purpose of this paper is to understand the fundamental requirements, challenges and opportunities associated with the opportunistic multi-player gaming in public transports, and our contribution is four-fold:

- We propose a completely new type of application scenario to enable highly interactive and opportunistic mobile gaming among nearby passengers in public transports, geared by the proposed CrowdMoG platform.
- We propose a high-level design of CrowdMoG to achieve the required functionalities, that include a few key design elements: a Peer Manager, a Game Session Manager, a Network Protocol Manager, a Group Manager, and a Game Feature Extractor.
- We explicitly describe various design challenges and difficulties for CrowdMoG, including the passenger dynamics, gaming group discovery, network latency issues and runtime game operation.
- We show experimental results related to the above barriers and opportunities that eventually provides insights to build such a gaming system.

The rest of the paper is organized as follows. First we describe the potential scenario which frames key challenges in building such a system. Then, we present the high level design prototype of CrowdMoG. After various challenges and possibilities is introduced, we conclude the paper and presents the future work.

II. POTENTIAL CROWDMOG OPERATION SCENARIO

We describe the motivating scenario for the considered opportunistic crowd-based gaming in public transports.

Patrick from France boards a metro train departing for the Louvre in Paris. Being a regular commuter, he knows that the journey will take approximately half an hour, and thus decides to kill the time by playing a racing game on his Android Tablet. With the in-situ gaming functionality supported by CrowdMoG, he is able to race with nearby passengers in the same metro car. Patrick strongly favors this type of location-aware mobile gaming since he could play with people in the real world [3] (however may not know exactly who he is playing with), and more importantly, he would not experience unexpected performance degradation while playing due to the network latency. When Patrick’s game starts, the CrowdMoG platform on his tablet disseminates the gaming information, including, but are not limited to, the preferred game type, future period of stay on the route, the number of required players and their levels, etc., to all nearby passengers. Assume that this information is proactively received by two passengers in the same metro car who have installed the same game in their Google Nexus. Note that in practice, passengers frequently change their location (e.g., getting on and off) while commuting, and thus the maximum allowed time window to join a game is relatively very short (as analyzed in Section Gaming Group Discovery). In such cases, just making an application layer protocol to probe for nearby peers is far inefficient, since it will result in high battery drain while continuously searching for peers. Using our situation-aware matchmaking process as part of our CrowdMoG system, it helps only identify the GOs of the ongoing games if they are of similar interests of Patrick’s (rather than finding all game participants). Furthermore, CrowdMoG also performs seamless session handoff rightly after Patrick’s departure so that other game participants who were gaming with him are still able to maintain a satisfactory continuity. Hence, the proposed system not only deals with intricacies related to the realtime on-the-fly gaming, but also it solves the fundamental research problem of how to form a game group opportunistically at runtime among collocated travelers.

In the following discussions, we similarly call a passenger behaving like “Patrick” as the GO, those travelers who are physically around Patrick and within the communication range as the “crowd”, and finally once they join Patrick in a game play, they are called the participants of a gaming group. A crowd may consist of multiple collocated simultaneous gaming groups.

III. CROWDMOG HIGH LEVEL DESIGN

Figure 2 shows a high level architectural design of CrowdMoG. It consists of four layers built on top of the smart device OS (e.g., Android’s OS or iOS for Mac products). The top layer uses the existing open source game engines like [12], responsible for I/O command processing, graphic rendering, and handling APIs. The next layer below consists of a handful of session APIs (where an example is shown in Table I) provided by the CrowdMoG. These APIs are linked with the Game Session Manager module (see Section Game Session Manager) in the Game Controller layer. It is primarily responsible for maintaining the game session through the Connection Manager as well as efficient peer discovery and selection by Peer Manager module (see Section Peer Manager). The session information is then passed to the Information layer, which maintains repositories to store the mobility pattern of passengers like the average commute time and route, and the Group Manager module to store any information related to the group (like its size and type;
A. Peer Manager

Finding peer-of-interest is the primary functionality of CrowdMoG platform on each smart device. Here, the Peer Manager module performs both the peer detection and selection, to associate the end user with all interested nearby participants, without being impacted by any side effect of high discovering latency. As part of the situation-aware matchmaking process, peer detection and selection are only performed by the GO, who initializes the game at the very beginning, or elected rightly after the originator’s departure, rather than finding all devices by each passenger individually. Details regarding the necessities of such discovery process will be explained in Section Gaming Group Discovery. Specifically, Peer Detector module is responsible for finding all relevant gaming groups in the surrounding, whereas the Peer Selector module only selects those with similar gaming preferences to the user, by input from the Game Feature Extractor in Section Game Feature Extractor. This flexible functionality enables every commuter to associate himself/herself only with their preferred game play, given the fact that potentially many different types of games co-exist in a crowd, and the size of the crowd can be in the scale of 50 people or more (during peak hours).

Furthermore, interacting with the Access Controller module, Peer Manager enables the group association only if the participants are mutually trustworthy and their size does not exceed a threshold (see Section Network Protocol Manager). It provides an opportunity in case a commuter favors to interact with whom he/she has played earlier in a game and makes sure the admitted new player will not impact the existing game performance. The degree of trusts can be quantized by parameters like average session length between two players.

B. Game Session Manager

Since our targeted application scenario is associated with a highly dynamic crowd, maintaining game sessions becomes extremely important. This module serves as part of the situation-aware matchmaking process, to provide passengers a unique way to attain continuous session management. Following Patrick’s example earlier, since he is a regular commuter, the CrowdMoG platform running on his smart device periodically records his travel pattern using the trip memory application [2], [13], which has been prototyped as a proof-of-concept. It is an Android application that tracks the commuter’s traveling path and logs the surrounding events extracted from the ambient sensors. The collected trip information (stored in the mobility trace repository of the Information layer) is then distributed to the GO, and the corresponding Session Manager module coordinates the session handoff well before his/her departure through Network Protocol Manager (see Section Network Protocol Manager). In this way, it facilitates the mobility pattern overhearing inside the group, and ultimately helps all participants achieve the continuity of the game play.

C. Network Protocol Manager

The main drawback of using 3G/4G-based cellular networks to facilitate the crowd gaming in public transports is due to its poor wireless link quality (more experimental will be given in Section Network Latency). This problem can be more severe in some cases where infrastructure is not available (like in the tunnels of London Underground). Thus, traditional client-server model is not feasible. Meanwhile, the broadcasting based protocols like WiFi-Direct [14] suffers from injecting significant amount of co-channel interference within a group, even under a reasonable size of 50 passengers (as a usual case during peak hours). Towards this end, our Network Protocol Manager aims to reduce the amount of packets exchanges in the crowd while improving the latency performance of the network in realtime.

Instead of communicating with the back-end game server for each game packet (e.g., mobility and game information), one possible solution is to operate the CrowdMog platform in a two-phase mechanism. Each participant unicasts the game packet to the GO in the first phase, and then the GO broadcasts the collected set of information to the crowd. In this way, the communication only happens within the local game group, and the GO is the only one to broadcast the packets, while overheard by all participants. Furthermore, unlike traditional prediction-based methods (e.g, interpolation [15] and extrapolation [16]) that estimate other characters’ movements by using their past information, our Network Protocol Manager accurately disseminates the gaming information among the game group. This overcomes the problem of cellular network connectivity, and helps distribute the exact gaming information among all participants, rather than using any prediction-based methods that aims to estimate their movements. This is feasible in that the size of the group remains relatively stable during the travel time between two stops.

Finally, since the size of the group highly impacts the performance of network overhearing, this module aims to dynamically estimate an optimal threshold, defined as the maximum number of participants in a group, beyond which the interference may deteriorate the protocol performance. This
threshold can be quantized by parameters like the received signal-to-noise-plus-interference-ratio (SINR) at runtime.

D. Group Manager

This module maintains the gaming group information like the type of the game being played and number of participants. Upon request, the GO’s Group Manager module distributes this information to the new passengers to aid their joining decision, which is eventually performed by the Peer Manager in Section Peer Manager. Also, this information will be updated when existing passengers leave the game. Note that it will be updated periodically at runtime, and we only store it in a memory-based device cache (without persistence guarantees). As discussed earlier, the size of the group is retrieved by the Access Controller module to impose a hard requirement on the maximum allowed number of participants admitted in the game, to provide satisfactory latency performance.

E. Game Feature Extractor

This module is primarily responsible to collect and analyze the smart device user’s gaming preferences, e.g., the preferred game type and role, the average duration of the game play, the preferred type of competitors, etc. These pieces of information can be either collected periodically at runtime through well-known online social networking websites like Facebook or Twitter, or whenever there is an Internet accessibility (e.g., when the metro is approaching the platform where the infrastructure networking facilities can always be assumed). Then, this module is expected to perform runtime analysis to extract the gaming preferences of the user, and later stored in the Game Preference repository in Information layer. In this way, the Peer Selector module can leverage it to initialize the situation-aware matchmaking process to identify the right group of passengers to play with. The fundamental research question to ask when performing multi-player gaming in public transports is how to dynamically form such gaming group from nearby passengers. However, it is very challenging due to the mobility of participating users (like where to get on/off and their physical movements inside the vehicle), and the heterogeneity of the smart devices (e.g., available communication protocols, platforms, energy reserve, etc.). Our first set of experiments aim to understand how dynamic the considered environment is.

We developed a Bluetooth-based neighbor discovery application in an Android smartphone, where a tester carries the handset traveling in two fixed routes of Beijing Underground everyday from June to September 2012. In total, we received 412,806 lines of records, each of which is composed of the timestamp of the record, the discovered Bluetooth MAC address, and the corresponding received-signal-strength-indicator (RSSI). It is worth nothing that we use Bluetooth mainly due to the legacy reasons that many old handsets do not support WiFi, however we shall discuss the applicability of the results if WiFi is considered for smart devices these days.

As shown in Figure 3, we first demonstrate the ECDF (empirical cumulative distribution function) of the speed of passenger flows, as a indicator to illustrate how dynamic the environment is as the commuters come and go over time. We observe that on average around 6.97 passengers coming in and out from the metro car per minute. That is, for one stop of two minutes in the route (as a typical operation in Beijing Underground), one can expect 14 new neighbors appearing in the surrounding. Given their diverse gaming preferences, this result implies that a passenger will have an ample of opportunity to start/join a game play in the crowd, while no need to worry about the followers.

Figure 4 describes the passenger lifetime statistics, also in ECDF. We define and compute their lifetime as the period of stay within the communication range of the tester (5-10 meters in Bluetooth Class 2 [17]), i.e., by time-stamping any MAC address when it appears for the first time and last time, we compute their difference. Note that it has nothing to do with the passenger’s journey time, but it indeed clearly indicates the stability of a game group, or how long a particular game may last. From the raw data, it is interesting to find that some MAC addresses only show once (perhaps due to the local movements inside the metro car), and thus, for those records, we opt to adopt two different processing methods. One is to assume that they only travel for one stop next to the tester (the worst case) and then leaves the train, or 120 seconds in the case of Beijing Underground, referred as “primary data”; while the other method is simply to treat them as noise, referred as “filtered data”. As shown in the figure, the average lifetime for the filtered data set is 470.91s, i.e., on average a passenger will stay with the tester for 3.9 stops, and for the primary data the mean value is 279.46s, or 2.3 stops. This further confirms that it is very rare that a commuter only travels for one stop then left. Furthermore, the median value of 225s (1.9 stops) indicates that 50% of all passengers one can meet on a route will travel approximately two stops together. Therefore, it is sufficient to conclude that time is not a problem for the considered FPS or racing game play.
B. Gaming Group Discovery

Given the fact that passengers in public transport scenarios are behaving dynamically and their average period of stay shines the possibility of playing multi-player games, the question still remains on how to efficiently detect the nearby collocated devices, especially when considering their associated gaming preferences. In delay tolerant networks (DTNs), each device aims to learn the mobility pattern of all nearby devices for information sharing [18], [19], however it is not applicable in our scenarios. Consider an example of a crowd of 50 people and the larger the size of the crowd, the better diversity and opportunity to match their gaming preferences with that of the arrival passengers who will join the game play. Unfortunately, learning the mobility pattern of such a big crowd is almost impossible, given the fact that the period of vehicle stay on one platform is just in the second scale (e.g., 20 seconds on average in Beijing Underground). One such example for a unicast-based peer discovery by using Wi-Fi Direct shows that maximally 10 users can be detected in a 15-second time window by using a 1MHz channel of 900MHz band [20]. Nevertheless, the amount of surrounding passengers far exceeds this number. The problem becomes even more severe, if further to complete the download of game session information (by Game Session Manager) and connection establishment, all within a time window of 20 seconds. Additionally, energy efficiency issues further prohibits the frequent scanning of nearby devices. It is obvious that none of the existing protocols can achieve this.

C. Network Latency

We investigate the difficulties of using cellular network as the candidate access technology. We perform an experiment where two Android phones periodically send test packets to the back-end game server through cellular connections while traveling in both buses and metro in Seoul. One phone is 3G enabled by SK Telecom and the other one is 4G LTE supported by LG U+. We record the round trip time (RTT) of each packet. Furthermore, the experiment is undertaken in non-peak hours that eventually serves as the baseline results (and it could be highly variable during different time of the day [21]).

Figure 5 shows the latency measurements in buses. We observe that 3G performs terribly with RTT reaching up to 28.27s, 738.518ms on average (standard deviation 3.147s) and 5% packet loss. The similar results have been obtained while using 4G, as shown in Figure 6. Although slightly lower than that of 3G, it still maintains an average of 128.735ms and standard deviation 124.940ms (with the maximum value up to 9.507s).

Figure 7 shows the experimental results of RTT in the metro by using the 3G network, where RTT achieves 221.885ms on average (standard deviation 209.067ms) and 7% packet loss, significantly lower than that of the bus case. This is particularly so because of the deployed “network boosters” by cellular network providers to sustain cellular signals [22], however it cannot be assumed anywhere in the metro of other metropolitan cities (an extreme case happens in London Underground without any cellular signal in the tunnel). Similarly, we obtain results by using 4G network in the metro, as shown in Figure 8. It shows substantially better performance with an average RTT of 97.312ms (standard deviation 47.013ms) and 0.5% packet loss, if compared with that of 3G. However, we still observe sudden communication loss and high peak delay (>1s) sporadically which may significantly deteriorate the multi-player gaming experience when traveling.

Since we are targeting multi-player games likes FPS and racing with latency requirements around 100-200ms and negligibly small jitter, 3G network cannot serve the purpose for satisfactory gaming experience at all, and 4G network’s...
unexpected high variance and high infrastructure costs may sometimes be a serious problem. Furthermore, it is very important to observe a sudden link loss happened in both 3G and 4G metro case as shown in Figure 7 and Figure 8, which may significantly impact on the overall gaming experience. The results further confirm that a local communication paradigm is strongly desired to go around the cellular issues.

D. Runtime Game Operation

As analyzed in the previous section, large network latency by employing the traditional client-server based model running on cellular networks prohibits the realtime exchange of gaming information among the participants. Thus, existing technologies opt to predict the other player’s (character’s) movements as a suboptimal solution. Exemplary technique is the dead-reckoning concept [16] (extrapolation-based) to estimate the new position of the player by predicting his/her trajectory through the most recent information update. Similarly, interpolation based techniques [15] also have been proposed. Nevertheless, due to the imperfect prediction results and thus the errors deviating from the ground truth, it will significantly affect the game play experience at runtime.

Recently, overhearing the broadcast packets have been applied in a typically tethered connection type, to support the traditional server-client model in a local video streaming [23]. It aims to overcome the shortages of prediction based techniques, but comes with shortages from the expense of heavy energy drain on 3G/4G, heavy co-channel inter-group interferences [23], and without the support of session handoff upon participant’s departure and new members joining in. These problems need to be clearly addressed in the Network Protocol Manager module.

V. CONCLUSION AND FUTURE WORK

Multi-player gaming on smart devices are increasingly catching attention, particularly in public transport environments where people are looking forward to playing rich contextual games while traveling. In this paper, we proposed CrowMog, a crowd-based mobile gaming platform to enable highly interactive and opportunistic mobile gaming in such scenarios. The goal is to make every smart device as a computing substrate, leveraging other mobile devices in vicinity to achieve a distributed on-line gaming experience in real life.

Specifically, in this paper we proposed a high level design of CrowMog platform with key design elements of a Peer Manager, a Game Session Manager, a Network Protocol Manager, and a Game Feature Extractor. Supported by many modules, the matchmaking process offers a unique functionality to identify and match the nearby passengers on the move according to their associated gaming preferences, and provides the smooth session handoff to enable the continuity of existing game plays when participants leave the game due to different mobility patterns. Then, we described the potentials, along with challenges and opportunities, including the passenger dynamics, gaming group discovery, network latency issues, and runtime game operation. Eventually, we hope that our investigations can open up the new dimension for the research community to redesign and examine a traditional problem, fundamentally transforming it into a new era of mobile gaming experience.

As for the future, we plan to propose a local communication paradigm by using the concept of network overhearing to overcome the network latency issues imposed by cellular networks. Then, we will finalize the detailed design of each CrowMog component, and implement it on a real smart device. Finally, we plan to perform real experiments and test beds on the overall concept in buses and metro, and evaluate its performance. A lot of work can follow.

ACKNOWLEDGMENT

Corresponding author: Chi Harold Liu. This paper is sponsored in part by National Natural Science Foundation in China (No. 6130179).

REFERENCES