PGTP: Power Aware Game Transport Protocol for Multi-Player Mobile Games

Bhojan Anand[‡], Jeena Sebastian[‡], Soh Yu Ming[‡], Akhihebbal L. Ananda[‡], Mun Choon Chan[‡], and Rajesh Krishna Balan[†]
[‡]National University of Singapore and [†]Singapore Management University

***Abstract*—Applications on the smartphones are able to capitalize on the increasingly advanced hardware to provide a user experience reasonably impressive. However, the advancement of these applications are hindered battery lifetime of the smartphones. The battery technologies have a relatively low growth rate. Applications like mobile multiplayer games are especially power hungry as they maximize the use of the network, display and CPU resources. The PGTP, presented in this paper is aware of both the transport requirement of these multiplayer mobile games and the limitation posed by battery resource. PGTP dynamically controls the transport based on the criticality of game state and the network state to save energy at the wireless network interface (WNIC) level with almost no degradation to the quality of the game play.**

I. Introduction

The latest smartphones with PC-like processing power and OS are taking cell phones beyond the next level into a wireless world limited only by our imagination. As the capability of the device increases, new applications has emerged. The growth in hardware facilities and application sophistication comes with a tight penalty on battery life time. In modern smartphones, the three main sources of power consumption are, 1) the CPU, 2) the display, and 3) the network interfaces. These resources are consumed at varying rate according to the type of application. Games, which account for more than 50% of current iPhone application downloads [10], tend to consume large amounts of CPU and display power in general — with multiplayer games also consuming large amounts of network interface power.

PGTP is designed for energy and bandwidth efficient network control. PGTP manages the packets sent through the socket based on the game state and network state to control the network interface for power management. Mobile games require multiple streams between client and server through a single association (using one socket pair) with different quality requirements: reliable ordered streams, reliable unordered streams, partially reliable streams and unreliable streams. Our protocol provides all these features while optimising the power and bandwidth utilisation without affecting the QoS. We have implemented our protocol on Android's Dalvik VM on the clients side and Ubuntu Linux platform on the server side and evaluated its performance over WiFi and 3.75G WCDMA-HSPA networks with set of Google-Android phones running an Andriod game. We have shown that our protocol conserves up to 40% of power and maintains comparable latency and throughput with the existing protocols.

The main contributions of this paper are as follows:

1) We analyse the transport requirements of multiplayer mobile games.
2) We present the design of the PGTP.
3) We provide a reference implementation of PGTP on Linux based platforms and evaluation of the protocol against existing protocols over different network technologies.

This work is supported in part by the Singapore Ministry of Education Academic Research Fund Tier 2 under the research grant T208B2301. Any opinions, findings, conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the granting agency, National University of Singapore or Singapore Management University.

978-1-4244-9799-7/11/$26.00 © 2011 IEEE

II. Related Work

There is a rich body of related work on Transport protocols in general. Our extensive survey shows that there is no prior work on power aware transport protocols for mobile games. As shown in Section III-A, games have unique transport requirements. In the following section we show conventional protocols do not address all these requirements.

UDP [8] is the lightest and low latency protocol. Though it is widely used by game developers, it needs to be supplemented with application level custom protocols for reliability and ordered delivery. TCP [9] on the other hand enforces reliable and ordered transport for all bytes at the expense of additional delay due to head-of-line blocking. Since TCP is byte-oriented, the receiver is responsible to reconstruct the message structure from the byte data. As most of a game's packets require unreliable transport, enforcing reliability adds unnecessary overhead. A TCP connection or association supports only one stream. The authors of [3] show that TCP has features that makes it unsuitable for online games. From the game traces, it is shown that 46% of the bandwidth was occupied by packet headers alone. They also found that congestion control and fast retransmission failed to operate correctly. Though DCCP [7] offers UDP like transmission with congestion control it does not offer reliable or ordered transmission required by games. SCTP [5], [11] satisfies most of the requirements (datagram oriented, multiple streams with different delivery methods, congestion control, flow control and priority processing), but it is basically a connection oriented protocol and offers unreliable service through individual packet marking. SCTP is highly suitable for a communication which needs reliable

delivery for most of its packets with few packets for unreliable delivery. Communication of game applications exactly have the opposite requirement: more unreliable packets and very few reliable packets. A typical SCTP packet with data and acknowledgement, takes at least 44 bytes overhead for SCTP headers. Since most game packets are less than 32 bytes, more than half of the packet contains non-data and this makes SCTP very inefficient. SCTP do not support intermittent connection failures which are common in mobile environments and also a required feature for conserving power. Custom modifications can be made but the potential amount of change and effort may warrant it impractical. If PR-SCTP [12] cannot send a packet before its lifetime expires, it is simply dropped. This is required behavior for online games. However, if the packet has been sent but not yet acknowledged, it will still be re-sent even if the retry time exceeds the packet lifetime. This is unnecessary for time-sensitive packets. For priority processing, pSCTP [5] assigns each stream a priority and SCTP sends Heartbeat chunks to periodically probe an idle stream. Since high priority game packets are usually sparse and infrequent, this introduces needless network traffic. PGTP uses game state to make intelligent decision on priority processing hence it improves quality of the game play.

III. Design Requirements

A. Transport Requirements of Mobile Games

Online games have unique transport requirements when compared to other real-time applications. Online games usually send small packets (32 bytes) in a consistent and relatively slow rate [3]. Online games have different types of packets with each type requiring different type of QoS. For example, in Quake3 [6], some packets are time sensitive (for eg. movement update). This means that a delayed packet may not contain useful information. If a newer packet is generated with the latest information, the delayed packet can be simply dropped. In-game chat messages can be sent unreliably and a dropped chat packet would not significantly impact the game. In contrast, game score updates can be delayed but they cannot be discarded. Packets containing game changing events (for eg. shooting) should be transmitted immediately with high priority. Table I gives general set of the packet types present in today's commercial games and their associated QoS requirements.

B. Design Considerations

Based on the requirements discussed above in Section III we have the following recommendations for the power aware game transport protocol. We have designed the PGTP protocol based on these recommendations.

- **Lightweight and low overhead**: It has been shown that 98% of game packets have a payload of less than 32 bytes [3]. Thus, the overhead incurred by a transport protocol in managing the connection should be proportionally small compared to the payload.
- **Datagram oriented**: Since a game packet is a complete piece of information by itself, it would make sense to send and receive using datagram instead of byte stream.
- **Provides a range of delivery methods or service qualities in a single connection**:
  1 *Unreliable channel*: packets that do not considerably affect the game if they are lost or arrived out-of-order.
  2 *Reliable channel*: critical packets whose delivery must be guaranteed.
  3 *Semi-reliable channel*: time-sensitive packets that have become obsolete can be dropped.
  4 *Ordered arrival*: critical packets whose order-of-arrival matters.
  5 *Unordered arrival*: critical packets whose order-of-arrival does not matter.
  6 *Priority processing*: important packets containing game changing events should be processed first.
- **Throttling of transmission speed and temporary disconnection**: The protocol should throttle the transmission based on game state and network state. Most importantly, it should have built-in support for temporary disconnection when the WNIC is put to SLEEP mode for power saving.
- **Socket API**: The protocol should provide an easy API for game developers.
- **Congestion and Flow Control**: Congestion and flow control are required to throttle the sending rate to avoid packet loss due to a congested network or an overloaded receiver.
- **Multi-homing**: Multihoming for automatic transport layer level redundancy and load balancing (essential need for today's massively multi-player games).

IV. Design of PGTP

PGTP is a flexible multi-channel transport protocol. It supports temporary disconnection for power management, thus 'flexible'. PGTP treats data at the message-level and provides transmission over unreliable, reliable-ordered, reliable-unordered and semi-reliable-unordered delivery channels. Unlike SCTP [5], [11], the default channel is unreliable and additional packet level marking is required for reliability and ordered delivery. The game can specify parameters to describe how each data should be sent. These parameters indicate the delivery channel, order-of-arrival, data priority and lifetime through the Socket API. Data is processed in a highest-priority first-come-first-serve basis. If the data is flagged for the unreliable channel, it is simply sent. For data using the reliable channel, the send queue blocks and retransmit until an acknowledgement is received. During this blocking period, if there are data in the queue that are bound for the unreliable channel, they will bypass the block and be sent. Before sending each data, PGTP checks the lifetime parameter and drops the data if it has expired. Multiple data can also be sent in a bundle to save bandwidth. During the 'connected' state, if the connection is idle, KEEPALIVE messages are sent periodically to probe its status.

A. Datagram Structure

The PGTP header is designed to be as 'slim' as possible to save on bandwidth. The common header requires only 10

TABLE I
TYPES OF PACKET AND CHANNEL QOS

Example	Type of Packet: Time sensitive	Critical	Order	Priority	Type of Channel
Important game changing event	Yes	Yes	Important	High	Reliable, realtime, ordered
Movement update	Yes	No	Important	Normal	Unreliable, realtime, ordered, prioritized
Score update	No	Yes	Not Important	Normal	Reliable, buffered, unordered
Chat	Yes	No	Important	Low	Unreliable

Time sensitive - Time sensitive packets should be dropped if delayed beyond expiry
Critical - Critical packets should not be dropped
Order - Packet's order of arrival at destination
Priority - Priority of packets; a higher priority packet will be processed first for delivery

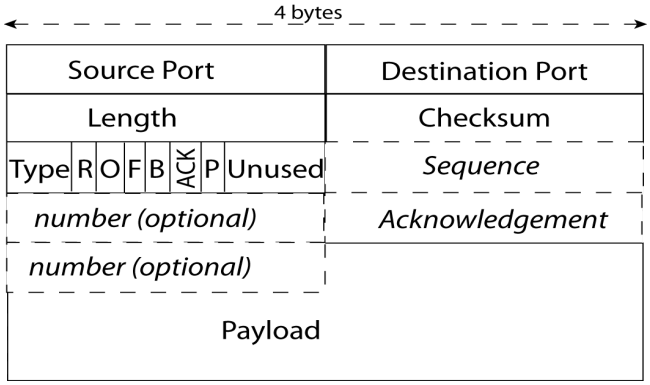

Fig. 1. Structure of PGTP datagram

bytes. The structure of PGTP datagram is shown in Figure 1. The fields **Source port** and **Destination port** fields identifies the processes communicating. The **Length** field holds is length of the message and the **Checksum** is one's complement of sum of 16 bit words. The **Sequence** field is optional and is added if the Reliable flag is set to 1. Similarly, the **Acknowledgement** field is added only if the ACK flag is turned on. No field is dedicated for payload length since this can be calculated by subtracting the header size from the packet length.

The **Type** field specifies the type of the packet takes one of the following values. *0000*: OPEN, *0001*: DATA, *0010*: KEEPALIVE, *1111*: CLOSE. There are five **Flags** of one bit each which have the following semantics. *Reliable (R)*: The reliability flag. When set to 1, this packet has a Sequence number and requires an acknowledgement from the receiver. *Ordered (O)*: The order-of-arrival flag. When set to 1, the receiver drops the packet if the Sequence number is not within the range of the receive window. *Fragmented (F)*: When set to 1, this packet contains only a fragment. When set to 0, this packet is the last or only fragment. The order of fragment is given by the Sequence number. *Bundled (B)*: When set to 1, it indicates that this packet contains more than 1 sub-packet. *Acknowledgement (ACK)*: When set to 1, this packet carries an Acknowledgement number. **Priority (P)**: This is a 3 bit field specifies the packet priority. A bigger number corresponds to a higher priority. **Unused**: The 4 bits unused field is reserved for future use (eg. game, power and network specific twists).

B. Connection Establishment

PGTP uses a 2-way handshake to establish a connection as depicted in Figure 2 . The Sender sends a packet with Type field OPEN to the Receiver. The Receiver acknowledges the packet by sending a packet with Type field OPEN and ACK field set to 1. If an acknowledgement is not received from the Receiver after some time, the OPEN packet is retransmitted. If the number of resends exceeds the maximum number of retries, the Receiver is deemed unreached and the connection fails. Note that no Sequence or ACK number is required when establishing a connection.

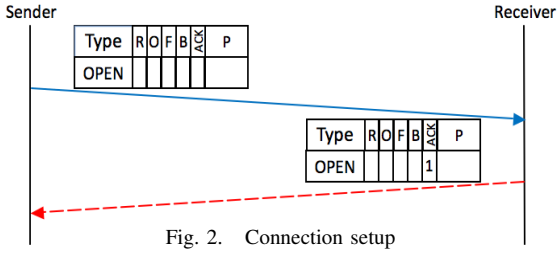

Fig. 2. Connection setup

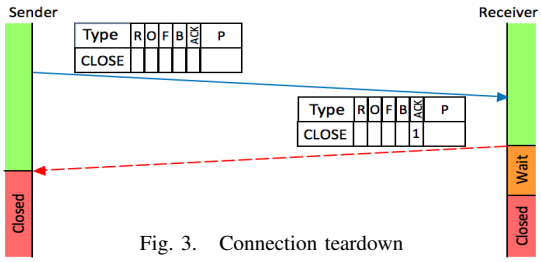

Fig. 3. Connection teardown

C. Connection Teardown

To close a connection, the Sender stops sending data and sends a CLOSE packet to the Receiver to wait for the acknowledgement as depicted in Figure 3. Upon receiving the CLOSE packet, the Receiver stops sending data and sends a CLOSE Acknowledgement packet. The Receiver then waits for a period of time before terminating. This wait period is to allow the Sender to retransmit the CLOSE packet if it fails to receive the CLOSE ACK packet. If a CLOSE packet is received during this wait period, the Receiver resets the wait period timer. When the Sender receives the CLOSE Acknowledgement packet, it terminates the connection immediately. Note that no Sequence or ACK number is required when closing a connection.

D. Data Packet Transmission

To send data, the Type is set to DATA and the appropriate flags are set, based on the parameters specified by the game. If the Reliable flag is set, the Sequence number field is added. Similar to TCP, this sequence number is used for identification purposes. But unlike TCP, this sequence number identifies a packet instead of a byte. The reliable packet is placed on a timer to be sent repeatedly until an acknowledgement arrives or if the maximum number of retries has been exceed. While the packet waits for acknowledgement, the send queue is scanned for data to be sent. Data requiring the reliable delivery channel are put on hold while data bound for unreliable delivery is removed from the queue and sent immediately. Unreliable data are not blocked because they do not need an acknowledgement and consequently do not need to be resent. This prevents unnecessary packet hold up.

If the ACK flag is set, the ACK number field is added. Like TCP, the acknowledgement number is used by the Receiver to notify the Sender that the reliable data has been received and the Sender can proceed to send the next data specified by the number. The reliable data received must have a sequence number that is expected by the Receiver. If the sequence number is unexpected, the Receiver resends the expected sequence number. The Sender receives the acknowledgement number and sends the expected data.

When data from the game is larger than the network's MTU, the data is split into fragments and sent separately. All fragments of data are sent reliably using different sequence numbers. All fragments except the last have the Fragment flag set to 1 while the last fragment's flag is set to 0. When the Receiver encounters the first packet with Fragment flag set to 1, that packet is buffered to wait for the remaining fragments. As more fragments arrive, the Receiver arranges them using their sequence number. When the Receiver encounters the last packet with Fragment flag set to 0, the fragments are merged and passed to the game.

E. Packet Bundling

For efficient use of bandwidth, many small packets can be bundled together and send as one big packet as depicted in Figure 4. A new field is added to the start of each sub-packet to denote its length. The common header is also added since each sub-packet may have varying transmission properties. However, only one Sequence number and one ACK number is needed. If any of the sub-packets have the Reliable and ACK flags turned on, the corresponding flags in the first header will be set regardless of the transmission properties of the first sub-packet. This is to denote the inclusion of the Sequence number and ACK number fields at the beginning. If the Reliable flag is set, the bundled packet is to be acknowledged as a whole. The Reliable, Bundled and ACK flags in the remaining common headers are then ignored. For unreliable packets, PGTP can save up to 44% of the original packet header size. For reliable packets, the saving is even greater at 50% because only one Sequence number is required.

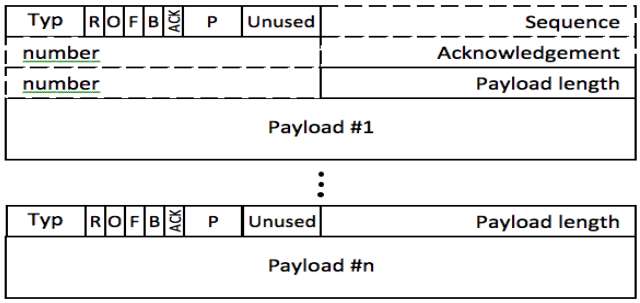

Fig. 4. Structure of Bundled Packet

Besides saving bandwidth, packet bundling also helps to improve throughput. When there are many reliable packets to be sent, the send queue is blocked until the current reliable packet has been acknowledged. By bundling them together, these reliable packets can be sent immediately, eliminating the time wasted on waiting for acknowledgement. This is particularly useful when the network interface is put into sleep mode. However, because the game is not aware of the switch off, it continues to send data to the transport protocol. This results in an accumulation of data in the send queue. When the network device is switched on again, there is a sudden burst of data to be sent. Without packet bundling, these data will take a longer time to be transmitted and this would cause undesirable delay jitter in the game. With packet bundling, the artificial delay is reduced.

F. Keep-alive

After the connection has been established, the Sender knows the Receiver is still reachable when it receives an acknowledgement for a reliable packet or when it receives a data packet. If the Sender only sends unreliable packet and the Receiver does not send any data, the Sender would not be able to detect a disconnected Receiver until it sends a reliable packet. As such, probe packets are sent periodically when the last reliable packet was sent too long ago. A KEEPALIVE timer is used to keep track of this duration. When the timer expires, a KEEPALIVE packet is sent and it waits for an acknowledgement. The timer is reset whenever a KEEPALIVE ACK or a normal ACK is received. If no acknowledgement is received after repeated sending, the connection is deemed 'broken'.

G. Resource State Aware Power and Bandwidth Optimisation

Unlike conventional transport protocols that only interpret the network to manage packet sending, PGTP factors in the state of the game and network to optimize power and bandwidth consumption for online games on mobile phones. PGTP operates in 4 modes that control how long packets are buffered for efficient use of bandwidth and power management. The modes are: *Optimal* (no buffering), *Good* (short buffering duration), *Normal* (medium buffering duration), and *Saving* (long buffering duration). The modes are determined by the criticality of the game state and quality of the network state (such as, RTT latency and Bandwidth) as shown in the Table II. The game or a middleware provides the information

TABLE II
DETERMINING THE POWER SAVE MODE

- - - - - - - Network State Game State	Good	Average	Bad
Non-critical	Saving	Normal	Good
Semi-critical	Normal	Good	Optimal
Critical	Good	Optimal	Optimal

about the criticality of the game state to the PGTP using the methods described in our previous works [1], [2] . In regular intervals PGTP computes the RTT latency by using the reliable transport stream and KEEPALIVE probes discussed earlier. The wireless node is put into SLEEP mode for the duration equivalent to the buffer duration of the mode to save energy with minimum or no loss of quality of service to the game application. For example, if the game state is **non-critical** and the network is **good** the PGTP switched to SAVING mode, and it will buffer the packets for a longer duration and save more energy. In general, buffering for a longer duration will increase the probability of expiry of a datagram. When a datagram has expired, it is not send but dropped. Since the datagrams are dropped during the non-critical state, it is not likely to cause significant disruption to the game.

V. RESULTS

A. Implementation

We implemented PGTP in *Google's Android* platform. The Android OS only supports Java applications and linked native C libraries. To achieve the highest performance, PGTP is implemented in C language and linked to a Java application using the Java Native Interface. The game communicates with the PGTP through Java method calls that are hooked up with the native C codes. To test the protocol we used *Armageddon*, a multiplayer mobile role-playing game created by our team with the features of the commercial role playing games.

B. Experiments

In the Armageddon game, each player controls an avatar that moves between game maps and attacks monsters. The game packets are recorded for a period of 5 minutes with 4 players connected to the game world through WiFi and 3.75G HSPA networks. From the traces, 2748 packets are collected for each client during the 5 minutes play test. Out of all the packets, 99.7% are sent unreliably and the remaining was sent reliably. 99% of the client packets were 6 bytes long. Most of the packets are sent at 110 to 120 ms interval. These characteristics are in line with the general online mobile game packet characteristics. As the results for WiFi and HSPA are similar, the results shown below are applicable for both networks.

As game updates are realtime and latency sensitive, most of the commercial multiplayer games use UDP with custom reliability protocols at application level. To find out how PGTP fare in comparison with UDP, we have collected similar traces for UDP protocol as well.

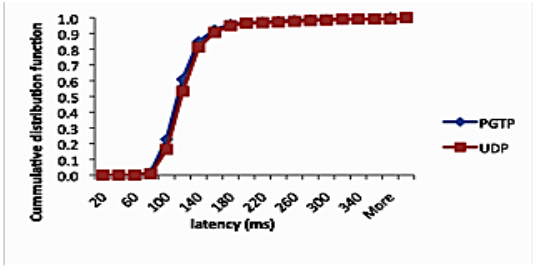

Fig. 5. Latency of PGTP protocol

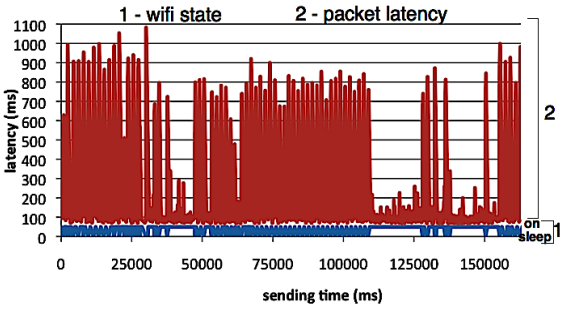

Fig. 6. Latency of PGTP Protocol with power management

C. Results: Base case - PGTP vs UDP

In this experiment, we consider three key parameters: latency, jitters and packet loss rates. PGTP is run in *vanilla mode* without packet bundling and power saving (WNIC is always ON). Despite, its ordered and reliable services, PGTP shows comparable performance in terms of latency (Figure 5) and jitter with UDP. The reason for PGTP's good performance can be attributed to the characteristics of the game packets and the implementation of the protocol. As mentioned earlier, 99.7% of the game packets are unreliable and thus the overhead incurred for reliable or ordered sending was minimal.

D. Results: PGTP with Power Management

Power is saved by putting the wireless interface into SLEEP mode when ever the game state is not important. This introduces additional artificial latency. In this experiment we look the (%) of power saved and its effect on latency and quality of the game play. PGTP is run *without bundling* feature. The results are shown in (Figure 6). Mean latency is 277ms (minimum latency is 57ms and maximum latency is 1082ms) and packet loss is 0.1%. As this latency happens only when the game state is not important, the game play is not adversely affected. A preliminary user study with 8 users of different levels of experience in games shows that the quality drop is almost not noticeable. Figure 7 shows that up to 40% of the power consumed by WNIC is saved with PGTP. Power saved is slightly sensitive to player density (number of players in a given game map). This is expected as increase in the number of players will increase the percentage of critical game states.

E. Results: Reducing PGTP Latency with Bundling

In this experiment we turn on both power management and packet bundling features of PGTP. Figure 8 shows that packet bundling significantly reduces the latency. The average latency

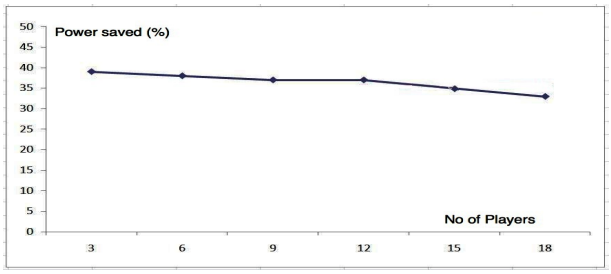

Fig. 7. Percentage of power saved

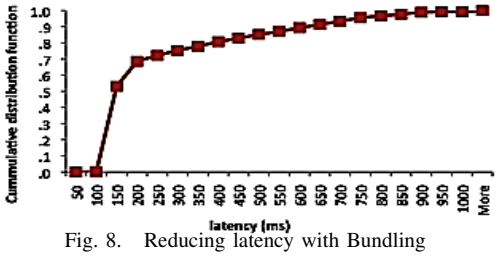

Fig. 8. Reducing latency with Bundling

is 249ms with 72.2% of packets with latency below 250ms. With bundling, PGTP experiences a higher packet loss (0.7%). The lost packets are denoted as 'More' in the figure.This is expected as a bundled packet is larger in size and is more likely to be dropped in a congested wireless networks. As we have capped the number of packets to be bundled to 5, the loss is minimal. Moreover, more than 60% of these lost packets are those dropped by PGTP in the send queue itself as their lifetime expired when they are in the queue. This is one of the expected behaviors of the protocol and the packets dropped are non-critical to the quality of the game play.

Figure 9 shows the correlation between the packet latency and the period of wireless on and off status. It is clear that, the latency experienced by majority of packets are caused by SLEEP mode of the WNIC. These delays are expected and they happen only during the game state is not-critical. Hence, the quality of the game play is assured.

VI. Limitations and Future Work

The current design and implementation is an *attempt towards a new protocol for mobile multiplayer games* which is efficient in terms of using power resources, bandwidth and reducing and/or hiding latency. We are extending the protocol further by designing and implementing the missing features. We have currently drafted multihoming for automatic transport layer level redundancy and load sharing facility in PGTP, which is widely required for massively multiplayer mobile games. Note, SCTP's multihoming is only for redundancy. Due to lack of space, we give a glimpse of the design for multihoming. *Redundancy:* A back-up IP is sent by the current server in the ACK packet during the connection setup phase (Figure 2). The challenging aspect here is transferring client game state to the new server. *Load balancing:* The current server picks the nearest server with less load and transfers the client's game state to it. It sends the IP of the new server to the client for hand-off as in GSM networks. Congestion control and flow control will be designed and implemented based on TCP friendly rate control [4] to keep the protocol

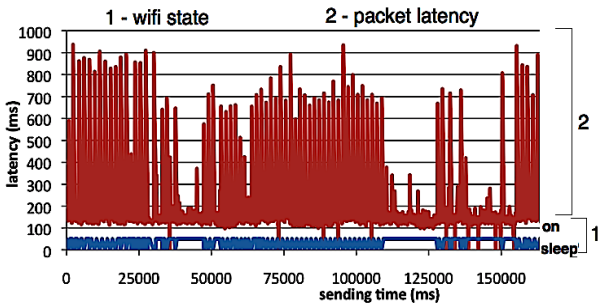

Fig. 9. Correlation between latency and WNIC state (SLEEP or ON)

simple and lightweight in comparison to TCP and SCTP. To avoid security attacks such as TCP-SYN attack, we have excluded sequence numbers from connection establishment and teardown packets. Sequence numbers can be securely negotiated after establishing connection. We will be designing and implementing security features for PGTP.

VII. Conclusion

The niche application area which PGTP addresses is *networked real-time interactive applications in battery operated mobile devices that have small payload sizes for transmission and do not require reliable transmission for most of their packets*. We have shown the basic design of the PGTP and its performance through experimental results. Through our protocol the mobile devices can save up to 40% of WNIC energy consumed games applications. We would like thank our lab and project members for their assistance and contribution in designing, implementing, and evaluating the protocol, associated game and the middleware.

References

[1] Anand, B. Konva: Power and network aware framework and protocols for multiplayer mobile games. *Proceedings of the Eleventh ACM SIGMOBILE Workshop on Mobile Computing Systems and Applications (HotMobile)*, Annapolis, Maryland USA, Feb. 2010.

[2] Anand, B., Thirugnanam, K., Long, L. T., Pham, D. D., Ananda, A. L., Balan, R. K., and Chan, M. C. Arivu: Power-aware middleware for multiplayer mobile games. *Proceedings of the Ninth IEEE NetGames*, Teipei, Taiwan, Nov. 2010.

[3] Chen, K.-T., Huang, C.-Y., Huang, P., and Lei, C.-L. An empirical evaluation of tcp performance in online games. *ACE '06: Proceedings of the 2006 ACM SIGCHI international conference on Advances in computer entertainment technology*, page 5, New York, NY, USA, 2006. ACM.

[4] Handley, M., Floyd, S., Padhye, J., and Widmer, J. Tcp friendly rate control (tfrc): Protocol specification, 2003.

[5] Heinz, G. J. Priorities in stream transmission control protocol (sctp) multistreaming. Masters Thesis Tech Report 2004-01, University of Delaware, Newark, DE, 2003.

[6] Id Software. *Quake 3 Arena Source Code*. http://ioquake3.org/.

[7] Kohler, E., Handley, M., and Floyd, S. Datagram congestion control protocol (DCCP). *Internet RFC 4340*, Mar. 2006.

[8] Postel, J. User datagram protocol. *Internet RFC 768*, Aug. 1980.

[9] Postel, J. Transmission control protocol. *Internet RFC 793*, Sept. 1981. Editor.

[10] Schonfeld, E. When it comes to iPhone games, what sells is action, adventure, and arcade. http://techcrunch.com/2010/02/25/iphone-games-what-sells-distimo/, Feb. 2010.

[11] Stewart, R. Stream control transmission protocol. *Internet RFC 4960*, Sept. 2007. Editor.

[12] Stewart, R., Ramalho, M., Xie, Q., Tuexen, M., and Conrad, P. Stream control transmission protocol (SCTP) - partial reliabity extension. *Internet RFC 3758*, May 2004.