Network Working Group
Request for Comments: DRAFT

Josh Willhite
June 2014

Internet Pong Protocol

Status of the Memo

This memo defines an Experimental Protocol for the Internet community. Discussions and suggestions for improvement are requested. Distribution of this memo is unlimited.

Abstract

The Pong protocol was developed as a class project for CS494 InternetWorking Protocols at Portland State University. This protocol is designed to allow Pong enthusiastes to connect and play games in real time. In this intial version of IPP (Internet Pong Protocol) design descisions have been focused on ensuring quality game play at the expense of a wide feature set. IPP could easily be expanded to include functionality for saving and resuming games, storing game results, and in game messaging.

Contents

1 Introduction

This document describes the Pong Internet Protocol. The protocol was developed over the course of several weeks as a class project for CS494 InternetWorking Protocols at Portland State University in May of 2014.

The protocol is designed to allow two Pong players to play real-time(or near real time depending on connection speed) games. In order to facilitate the real time game play the Pong Protocol is built on UDP/IP. UDP was chosen as appose to TCP to allow lower overhead and more flexibility at the cost of guaranteed packet delivery.

The Pong system makes use of the client-server model, as well as the P2P model. The server is used to connect players(clients). Once players are connected the only further communication to the server occurs when the players close the application. Game play is achieved through direct communication from one players machine to another's (P2P).

Future protocols may extend the servers role to include storing game results and ranking players, as this would necessitate implementing authentication and non-ephemeral storage on the server it is beyond the scope of this first protocol version.

1.1 Servers

Servers are used exclusively to facilitate P2P connections between individual Players. When queried by a Player looking to start a new game the server returns a list of other Players who also want to start a game. When two players agree to start a game they send a message to the Server which then removes them from the list of potential opponents.

1.2 Players

Players are software instances of the P2P client/server that implements this protocol on individual gamers machines. Note that a single end user may be running multiple instances on a single machine using different ports, this is perfectly fine and doesn't violate the RFC guidelines. (Although it may be tough to keep up with multiple games simultaneously :))

2 The Pong Specification

2.1 Overview

The protocol described here if for use with both Server connections and Player connections. Since Servers are just used to find opponents the majority of the protocol is for P2P communication. Messages are the fundamental unit of communication and are exchanged between hosts(Server ←→ Player, Player ←→ Player).

2.2 Messages

Messages are composed of three parts: source, action/response, and parameters. The "source" specifies the source of the message and can either be a Server or Player instance name. The action/response specifies the desired action or response, and the parameters which are optional contain the actual information. For instance a Player(source) could query(action) the server for a list of available opponents, the Server(source) would respond(response) with a list(parameters) of available opponents.

Depending on the action the host may or may not directly reply to the message. For example during game play Player instances send each other messages regarding changes to the game state. These messages are not directly replied to.

2.2.1 Message format in pseudo BNF

The message format structure is based on standard JSON format. This should allow easy serialization/deserialization since most popular programing languages have libraries which support working with JSON. In Python specifically it's possible to deserialize JSON directly into native Python data structures.

$$\begin{aligned}
< message > &::= "\{" < source > "," < action > "," 0 * 1 < parameters > "\}" \\
< source > &::= "SERVER"|"PLAYER"" : " < name > \\
< name > &::= 4 * 16(DIGIT/ALPHA) \\
< action > &::= "ACTION" : 1 * 16(DIGIT/ALPHA) \\
< parameters > &::= "\{"1 * (< key > " : " < value >)"\}"|"["1 * (< value >)",""]" \\
< key > &::= 1 * 16(DIGIT/ALPHA) \\
< value > &::= 1 * 16(DIGIT/ALPHA)
\end{aligned}$$

For example:

```
{"SERVER":"0.0.0.0:55556", "ACTION":"LIST", ["player1", "player2", "player3"]}
```

3 Pong Concepts

This section explains the reasoning behind the protocol design choices and provides an in depth view of the issues faced when trying to synchronize game state across the Internet.

3.1 Player to Server Communication

As described in the introduction section the Server exists solely as a means to initiate game connections with opponents. In order to accomplish this goal the server needs to maintain a list of available opponents and there associated IP address and port number. Basically it needs to be able to do four things: add a player, remove a player, provide the current opponent list, maintain the list.

3.1.1 Add Player

In order to add themselves to the list of available opponents on the server, a Player would send a message with the "ADD" action. The server then checks to see if the source name the Player used is unique. If it isn't unique the server responds with "ADDFAIL" and a list of the current opponents. Once the Player has chosen a unique name the server adds it to the list.

3.1.2 Remove Player

The final step in the initialization process between two Players occurs when they each send a "REMOVE" action to the server. Upon receipt of the remove, the Server takes them both off the available opponents list.

3.1.3 List Opponent's

In order to get an updated list of opponents a Player can issue a "LIST" action message to the server. The server of course responds with a "LIST" of available opponent's.

3.1.4 Maintain List

When a player drops off line without issuing a "REMOVE" action there needs to be a mechanism for taking them off the list. Once every 2 seconds the Server issues a "HEARTBEAT" action to all Players on in the list. If they fail to respond with a "BEAT" within 5 seconds then they are automatically removed.

3.2 Player to Player Communication

A game session life cycle consists of an initial handshake followed by game play and concluded when either a player has reached a score of 5 goals and won or a player leaves the session (either through connection timeout or withdraw from the game). Setting up and concluding the game session are relatively straight forward, the challenge lies in maintaining consistent game state between Players in as close to real time as possible.

3.2.1 Initialization

Initializing a game starts when Player-A chooses an opponent from the list and sends a "GAMESTART" action message to Player-B. Player-B can respond with "GAME" to start the game, or if no response is returned within a given amount of time Player-A assumes that Player-B is off line.

In the event of a "GAME" response first both Player-A and Player-B send a "REMOVE" message to the Server. Then game play begins immediately.

3.2.2 Game Play

Game play consists of the two Player instances A and B sending "UPDATE" messages back and forth in an un-synchronized non-blocking fashion. Once initialization has completed Player instance A randomly chooses a direction for the ball which always starts in the middle of the court, it then sends the first "UPDATE" message which Player-B is waiting for. Update messages consist of the following information:

1 Source: This is the name that the Player uses to register with the Server.

2 Action: "UPDATE"

3 Parameters: Ball Position, Ball Direction, Paddle Position, Goals, Update Number.

Since UDP offers no guarantee of in-order delivery the update number parameter shown above is used. If the current update number an instance has in memory is greater than the number of the update just received then it will ignore the just received update.

3.2.3 Player Lag

A ball can remain in play without any "UPDATE" messages from the opposing player until it reaches the opponent's side. At which point it will pause and wait for an "UPDATE" message, if no update is received within a set timeout the game is terminated.

3.2.4 Game Conclusion

The game concludes when one of the players achieves a score of 5 goals.

4 Message Details

4.1 Server Messages

4.1.1 ADD

$$\begin{aligned} Source &: PLAYER : "PlayerName" \\ Action &: ADD \end{aligned}$$

A Player can add themselves to a server's opponents list.

4.1.2 REMOVE

$$\begin{aligned} Source &: PLAYER : "PlayerName" \\ Action &: REMOVE \end{aligned}$$

Once game initialization has completed two opponents issue a REMOVE to the server to have their names and address removed from the opponents list.

4.1.3 LIST

---PLAYER QUERY---

$$\begin{aligned} Source &: PLAYER : "PlayerName" \\ Action &: LIST \end{aligned}$$

---SERVER RESPONSE---

$$\begin{aligned} Source &: SERVER : "ServerName" \\ Response &: LIST \\ Parameters &: \{"player1" : "0.0.0.0 : 00000", "player2" : "1.1.1.1 : 11111", ...etc\} \end{aligned}$$

The LIST message serves as both a query request from a Player and a response from the Server.

4.1.4 HEARTBEAT

---SERVER QUERY---

$$\begin{aligned} Source &: SERVER : \text{"}ServerName\text{"} \\ Action &: HEARTBEAT \end{aligned}$$

---PLAYER RESPONSE---

$$\begin{aligned} Source &: PLAYER : \text{"}PlayerName\text{"} \\ Response &: BEAT \end{aligned}$$

The HEARTBEAT message is used by the Server to keep the opponents list up to date. Every X number of seconds the Server issues a HEARTBEAT to all Players in it's list. If the Players fail to respond with a BEAT within 10 seconds then they are removed from the LIST.

4.2 Player Messages

4.2.1 GAMESTART

---PLAYER A QUERY---

$$\begin{aligned} Source &: PLAYER : \text{"}PlayerName\text{"} \\ Action &: GAME \end{aligned}$$

---PLAYER B RESPONSE---

$$\begin{aligned} Source &: PLAYER : \text{"}PlayerName\text{"} \\ Response &: GAMESTART \end{aligned}$$

GAMESTART is issued by a player in response to a GAME request, if the player responds with GAMESTART then the opponent can start the game. If however no response is recieved the player who issued the GAME request assumes the request is denied.

4.2.2 GAME

---PLAYER A QUERY---

$Source : PLAYER : "PlayerName"$
$Action : GAME$

GAME is issued by a player to another player when requesting to play a game.

4.2.3 UPDATE

$Source : PLAYER : "PlayerName"$
$Action : UPDATE$
$Parameters : \{"BPos" : [x, y], "BDir" : Angle, "PPos" : Y, "UNum" : Z, "LEFTSCORE" : T, "RIGHTSC$

The vast majority of messages exchanged while using this protocol will be UPDATEs. UPDATEs are used to maintain game state between opponents. Fortunately only 5 pieces of information are needed to completely describe the state of each Players game: current ball position, current ball direction in the form of an angle (1-360), current paddle position, current update number, and the number of goals that have been scored on the Player who sent the message.

The update number is used to avoid jitter caused by UDP's failure to guarantee in-order delivery of packets. If a Player receives an update that has an update number less then the current number than it's ignored.