Distributed Multiplayer Video Game

Ryan Wickman
University of Memphis
Memphis, USA
rwickman@memphis.edu

***Abstract*—**

I. Introduction

In this section I will provide a brief overview of the project and its implementation details.

A. Goal

The goal of this project initially was to create a full peer-to-peer multiplayer video game. However, due to time constraints, I switched the scope of it to mainly focus on the matchmaking Server. Thus my goal was updated to establish a good baseline matchmaking Server that players of a Video Game could use to find and join a game session. Although this was my goal, I still did work on other components as time allowed, details of which I will give in the next section and throughout this paper.

B. Overview

In this project, I worked on a few components of what make up a mutliplayer video game. What this entailed was making a way for users to send information amongst one another once in the game, a matchmaking server users could use to connect to a game, and the game itself. I decided to use a peer-to-peer architecture for the in game communication for users. While this may cause more latency than a dedicated, centralized server, it scales better and is more cost efficent for myself. In this, one player is chosen to host the game and act like the server. The rest of the players will communicate through the host user as if it was a dedicated server itself. They will use UDP to communicate as the game is a real-time application and is thus time sensitive. On the other hand, the matchmaking server will be centralized as there needs to be a single point for all users to connect and express interest in finding a game session to join. The users will use TCP, as opposed to UDP, to connect to the matchmaking server as reliable data transfer is important for finding and joining a game. The game was developed using the Unity game engine. It is has a main menu that users can use to find a game through the matchamking server or connect to a game directly by using an IP address of the host. The actual gameplay is a first-person sword fighting game. While I have a good working example for these components, this is far from the final product it will eventually be. Thus throughout this paper I will provide incite to where I beleive the application could be improved or expanded on.

II. Matchmaking Server

The majority of the time I spend on this project was focused on the matchmaking server. It ended up having a lot more moving components than I first anticipated. I used C++ to code everything and CMake to manage the build procces, and I used Boost.Asio to provide asynchronous networking capability. The server is made up of a few parts: matchmaking server interface, TCPConnection, and game queue.

A. Mathmaking Server Interface

This is the point in which the user will first connect to the server. The server is listening on a specified port and accepts incoming requests when the arrive. It asynchronously accepts the requests then initiates a callback that handles setting up the TCP socket to the client. This allows for multiple users to connect to the server at once. The socket is constructed by initializing a TCPConnection object that will handle the rest of the users communication to find a game. While I did consider creating a new thread for each connection, I did not due to time constraints and not wanting to deal with the complexity of adding multithreading such as locking resources, race conditions, ect.. When I update this application in the future I will potentially add this feature.

B. TCPConnection

The bulk of my time that I spent on the matchmaking server was on the TCPConnection class. This is due to many factors such as redesigns, updating call sequences, callbacks, and simply just the overall complexity of this component. The job of this component pretains to handling the communication with the client that allows them to connect to a game. It does so by sending packets of data that are detailed in particular format. The data in the packets are in JSON data format, which uses key-value pairs. This has to be serialized before it is sent over the network and deserialized when it arrives to its destination. This works especially well since the data exchange is between the Video Game which is programmed in C and the server which is programmed in C++ and there are libraries in both that support the use of this data format. The packets themselves were designed to be lightweight and easily extensible. While there are a many different packet types I created throughout this project, they all inherit from the parent class Packet. The Packet class has members for the header length, maximum body length, body length, a char array to store the data, packet type, functions to access these members, functions to decode and encode the header, and functions to encode and decode the body. The functions to encode and decode the body are pure virtual as to enforce the children classes to provide their own implementation to encode and decode the body. The reason for doing this is because the information contained in the body of a packet differs between packet types. While both the header and the body of a packet are stored in the same char array, they are separate entities. The header is what contains the amount of bytes the body of the packet holds. This is used to send variable sized packets and so that the receiver of a packet knows exactly how many bytes to read. The header, however, is always the same size. It currently is only 8 bytes however this and the maximum body length can be changed depending on the requirements of an inheriting child type.

Now, after talking about the data format it uses for interoperability, I will begin explaining the actua job of the TCPConnection class. After the client has established a TCP connection with the Matchmaking server, a TCPConnection object is instantiated. Then the first thing the client will do is send a FindGamePacket. This is one of the packet types that inherit from the Packet superclass. All of the subclass types will have the naming scheme of "XXXPacket". The FindGamePacket object contains additional members for a unique identifier for the client and the game type the user wants to join. This may change in the future to include more information, but this suffices for now. Once the server begins to recieve the data for the FindGamePacketin an asynchronous read operation, it will first only read 8 bytes for the header. Then, if this is successful, it will decode the header and read the amoount of bytes specified for the body. This is done by calling a function from the callback function that reads the header of the packet. This function called "TCPConnection::do_read_find_game" will initiate another asychronous read operation to read the body. Then once it is done reading the body, it will fire a callback that will add the user to a queue for the specified game type. The way the user is pushed on the queue is by first getting a shared pointer to the queue from a queue manager. Next, it will create a User object. Finally, the User object is pushed on the queue.

C. Game Queue

Before a user is able to join a game session, they need to wait their turn so that everyone who came in before them has the chance to join a game. This is why I went with a FIFO data structure such as a queue to implement this feature. However, in a video game with multiple game types, a user may only want to join a specific one. Thus, it would not make sense for users looking for different game types to be in the same queue. My solution to these requirements is to have a GameQueue superclass that is inherited from by all queues handling the users of a specific game type.

This class has members variables for the queue that stores weak pointers to Users, an unordered user map, the game type of this queue, the minimum and maximum size of a game session, the current queue size, and a boolean flag for if a game is currently getting prepared. The queue is a std::queue that simply stores std::weak_ptr¡User¿ types. The reason it stores this type and not a User directly is because a connection may get dropped while a user is in queue. If this occurs, then the queue will be able to correctly verify this by checking if the User object the pointer points to is expired before it uses it. The user map is used to keep track of the amount of times a user has been added to a queue. This, however, is not to prevent duplicate entries in the queue, but to make sure the user's place in the queue is updated. For example, once a game is initiated and a user is popped from the queue it will need to verify its count is equal to 1 to add it the game session. If its count in this maps is greater than 1, then it will decrement this count and go on to the next user in the queue. If its count is 0 then it assumed this user was erased from the queue and it does nothing. You might be thinking what is the point of all this and why not just remove the user from the queue when a duplicate is detected or if an erase is called? Initially I did think about doing this. However, for many reasons I decided against it. For one, on average it takes O(1) time to look up an element in a map, while it takes O(n) to erase an element from a queue. So it is faster. Albeit, it does add extra space complexity of O(n) so there is a bit of a trade off. Additionally, in the future when introduce multithreading to this application, race conditions will begin to be an issue. If I was to erase an element from the queue while concurrently trying to pop an element out undefined behavior may occur. To prevent this, the queue would have to be locked until the erase finishes. Instead a simple update to the user map will not effect the queue operation. The worst case scenario is a user is added to a game before they should be because they were removed from the queue before the user map could update their count. This of course could also be fixed by locking the queue, but would take less time on average due to the smaller time complexity. The current queue size is used to keep track of the amount of unique users in the queue. This will always be less than or equal to the actual queue size as there could be duplicate users in the queue. The game type of the game queue is represented as an enum value. I have a file, game_type.hpp, specifying all the possible game types as an enum.

There are also several member functions in the GameQueue class: push, pop, erase, prepare_game, and start_game. They are all pure virtual function as it is up to the inheriting class to define their behavior. This of course makes the GameQueue class abstract, and thus I only have the GameQueue header and not associated cpp implementation file. Currently, due to time constraints, I only have one game type called deathmatch. Its class is aptly named DeathmatchGameQueue. All future GameQueue children will follow this naming convention of "XXXGameQueue". This class has same the members as the GameQueue superclass, albeit, with its own unique implementation of member functions of which I will explain in depth.

The DeathmatchGameQueue::push function has a single parameter of type std::shared_ptr¡User¿ aptly named user. This is what is being requested to be added to the queue. First, it will see if the user has already been added to the queue by checking if the user ID is in the user map. If it is not, it is inserted in the user map with a count of 1 and increment the queue size. If it is in the queue, but its count is less than 0 meaning its been erased from the queue but not yet popped off, then it will set its count to 1 and increment the queue size. If it is already in the queue and its count is greater than 1 then it will only increment its count. After checking its count, the user is added to the queue. Finally, if with this added users queue size made it large enough to start a game and a game is already not getting prepared it will call DeathmatchGameQueue::prepare_game.

The DeathmatchGameQueue::pop function is a little more complicated than I initially hoped. This is due to the unforeseen complexity of having duplicates in the queue and the potential of having a user whose connection is closed in the queue. When the pop function is called, it will first check if the current queue size is less than or equal to 0. If it is, it will return a default weak_ptr that points at nothing, so pretty much just null. Then, if queue size is greater than 0, it will keep trying to get a user off the queue until it find a user that still has an active connection. If no such user is found it will return weak_ptr¡User¿(). Now, once a user is found, it will check if the user is duplicated in the queue. If it is, it will pop out users out of the queue until it finds one that is not duplicated. When duplicates are removed in this way, their count is decremented. Finally, once a nonduplicated, alive user is found it will erase it from the user map, decrement the queue size, and return a weak pointer to the user.

The DeathmatchGameQueue::erase function is rather simple. It first checks if the user is in the user map. If it is, then it is removed from the user map and the queue size is decremented. This operation is again O(1) on average rather than O(n) and thus the erase function becomes O(1) on average.

The DeathmatchGameQueue::prepare_game function is what I said earlier is called once the queue reaches a certain minimum size. Firstly, it will set a boolean flag to true to indicate a game is getting prepared. Secondly, the user in the front is popped off the queue. Thirdly, it will call the host_callback function on the user passing in a reference to DeathmatchGameQueue::start_queue function and the game type of this queue.

The DeathmatchGameQueue::star_game function is used as a callback that is fired by the TCPConnection object of a host of a game once it is ready for users to join the game it is hosting. It has a single parameter for a JoinPacket. First, the packet is encoded, which serializes the packet to get ready to be sent over the wire to the user. Then, the it will enter in a while loop that will continue as long as a maximum amount of users have been added or the queue is empty. Next, inside this loop a user is popped of this queue, the current game size is incremented, and the join_callback on the user is called with the JoinPacket object passed as an argument. Finally, the preparing game flag is set to false.

Now that wraps up the implementation and design of the game queue.

D. User

The User class was created so that an item could be added to the game queue that is representative of a unique user. Additionally, it contains smart pointers that contain references for callback functions that are used to allow the user to join or host a game. Originally I planned on having a reference to the TCPConnection object in the User class. However, the problem to this is it would create a circular dependency with the GameQueue class. This is because the GameQueue class includes the header for the User class declarations and the TCPConnection includes the header for the GameQueue class declarations. So, instead the User has reference to the functions ”TCPConnection::host_game and ”TCPConnection::host_game” to prevent such a circular dependency, but still have the necessary functionality. This also provides more control for the TCPConnection to decide and handle what exact functions gets called. In the future, this may prove helpful if I require different users to behave differently when joining or hosting a game. One such instance would be if a private game is being created where only select users are able to join. This join callback could provide additional authentication statements to verify a user can join a game before making a failed attempt and the TCPConnection closing thinking the user found a game.

III. Prepare Your Paper Before Styling

Before you begin to format your paper, first write and save the content as a separate text file. Complete all content and organizational editing before formatting. Please note sections III-A–III-E below for more information on proofreading, spelling and grammar.

Keep your text and graphic files separate until after the text has been formatted and styled. Do not number text heads—LaTeX will do that for you.

A. Abbreviations and Acronyms

Define abbreviations and acronyms the first time they are used in the text, even after they have been defined in the abstract. Abbreviations such as IEEE, SI, MKS, CGS, ac, dc, and rms do not have to be defined. Do not use abbreviations in the title or heads unless they are unavoidable.

B. Units

- Use either SI (MKS) or CGS as primary units. (SI units are encouraged.) English units may be used as secondary units (in parentheses). An exception would be the use of English units as identifiers in trade, such as “3.5-inch disk drive”.
- Avoid combining SI and CGS units, such as current in amperes and magnetic field in oersteds. This often leads to confusion because equations do not balance dimensionally. If you must use mixed units, clearly state the units for each quantity that you use in an equation.
- Do not mix complete spellings and abbreviations of units: “Wb/m^2” or “webers per square meter”, not “$webers/m^2$”.

Spell out units when they appear in text: ". . . a few henries", not ". . . a few H".
- Use a zero before decimal points: "0.25", not ".25". Use "cm^3", not "cc".)

C. Equations

Number equations consecutively. To make your equations more compact, you may use the solidus (/), the exp function, or appropriate exponents. Italicize Roman symbols for quantities and variables, but not Greek symbols. Use a long dash rather than a hyphen for a minus sign. Punctuate equations with commas or periods when they are part of a sentence, as in:

$$a + b = \gamma \tag{1}$$

Be sure that the symbols in your equation have been defined before or immediately following the equation. Use "(1)", not "Eq. (1)" or "equation (1)", except at the beginning of a sentence: "Equation (1) is . . ."

D. LaTeX-Specific Advice

Please use "soft" (e.g., `\eqref{Eq}`) cross references instead of "hard" references (e.g., `(1)`). That will make it possible to combine sections, add equations, or change the order of figures or citations without having to go through the file line by line.

Please don't use the `{eqnarray}` equation environment. Use `{align}` or `{IEEEeqnarray}` instead. The `{eqnarray}` environment leaves unsightly spaces around relation symbols.

Please note that the `{subequations}` environment in LaTeX will increment the main equation counter even when there are no equation numbers displayed. If you forget that, you might write an article in which the equation numbers skip from (17) to (20), causing the copy editors to wonder if you've discovered a new method of counting.

BibTeX does not work by magic. It doesn't get the bibliographic data from thin air but from .bib files. If you use BibTeX to produce a bibliography you must send the .bib files.

LaTeX can't read your mind. If you assign the same label to a subsubsection and a table, you might find that Table I has been cross referenced as Table IV-B3.

LaTeX does not have precognitive abilities. If you put a `\label` command before the command that updates the counter it's supposed to be using, the label will pick up the last counter to be cross referenced instead. In particular, a `\label` command should not go before the caption of a figure or a table.

Do not use `\nonumber` inside the `{array}` environment. It will not stop equation numbers inside `{array}` (there won't be any anyway) and it might stop a wanted equation number in the surrounding equation.

E. Some Common Mistakes

- The word "data" is plural, not singular.
- The subscript for the permeability of vacuum μ_0, and other common scientific constants, is zero with subscript formatting, not a lowercase letter "o".
- In American English, commas, semicolons, periods, question and exclamation marks are located within quotation marks only when a complete thought or name is cited, such as a title or full quotation. When quotation marks are used, instead of a bold or italic typeface, to highlight a word or phrase, punctuation should appear outside of the quotation marks. A parenthetical phrase or statement at the end of a sentence is punctuated outside of the closing parenthesis (like this). (A parenthetical sentence is punctuated within the parentheses.)
- A graph within a graph is an "inset", not an "insert". The word alternatively is preferred to the word "alternately" (unless you really mean something that alternates).
- Do not use the word "essentially" to mean "approximately" or "effectively".
- In your paper title, if the words "that uses" can accurately replace the word "using", capitalize the "u"; if not, keep using lower-cased.
- Be aware of the different meanings of the homophones "affect" and "effect", "complement" and "compliment", "discreet" and "discrete", "principal" and "principle".
- Do not confuse "imply" and "infer".
- The prefix "non" is not a word; it should be joined to the word it modifies, usually without a hyphen.
- There is no period after the "et" in the Latin abbreviation "et al.".
- The abbreviation "i.e." means "that is", and the abbreviation "e.g." means "for example".

An excellent style manual for science writers is [7].

F. Authors and Affiliations

The class file is designed for, but not limited to, six authors. A minimum of one author is required for all conference articles. Author names should be listed starting from left to right and then moving down to the next line. This is the author sequence that will be used in future citations and by indexing services. Names should not be listed in columns nor group by affiliation. Please keep your affiliations as succinct as possible (for example, do not differentiate among departments of the same organization).

G. Identify the Headings

Headings, or heads, are organizational devices that guide the reader through your paper. There are two types: component heads and text heads.

Component heads identify the different components of your paper and are not topically subordinate to each other. Examples include Acknowledgments and References and, for these, the correct style to use is "Heading 5". Use "figure caption" for your Figure captions, and "table head" for your table title. Run-in heads, such as "Abstract", will require you to apply a style (in this case, italic) in addition to the style provided by the drop down menu to differentiate the head from the text.

Text heads organize the topics on a relational, hierarchical basis. For example, the paper title is the primary text head because all subsequent material relates and elaborates on this one topic. If there are two or more sub-topics, the next level head (uppercase Roman numerals) should be used and, conversely, if there are not at least two sub-topics, then no subheads should be introduced.

H. Figures and Tables

a) Positioning Figures and Tables: Place figures and tables at the top and bottom of columns. Avoid placing them in the middle of columns. Large figures and tables may span across both columns. Figure captions should be below the figures; table heads should appear above the tables. Insert figures and tables after they are cited in the text. Use the abbreviation "Fig. 1", even at the beginning of a sentence.

TABLE I
TABLE TYPE STYLES

Table Head	**Table Column Head**		
	Table column subhead	***Subhead***	***Subhead***
copy	More table copy[a]		

[a]Sample of a Table footnote.

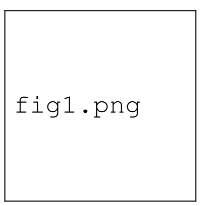

Fig. 1. Example of a figure caption.

Figure Labels: Use 8 point Times New Roman for Figure labels. Use words rather than symbols or abbreviations when writing Figure axis labels to avoid confusing the reader. As an example, write the quantity "Magnetization", or "Magnetization, M", not just "M". If including units in the label, present them within parentheses. Do not label axes only with units. In the example, write "Magnetization (A/m)" or "Magnetization {A[m(1)]}", not just "A/m". Do not label axes with a ratio of quantities and units. For example, write "Temperature (K)", not "Temperature/K".

ACKNOWLEDGMENT

The preferred spelling of the word "acknowledgment" in America is without an "e" after the "g". Avoid the stilted expression "one of us (R. B. G.) thanks ...". Instead, try "R. B. G. thanks...". Put sponsor acknowledgments in the unnumbered footnote on the first page.

REFERENCES

Please number citations consecutively within brackets [1]. The sentence punctuation follows the bracket [2]. Refer simply to the reference number, as in [3]—do not use "Ref. [3]" or "reference [3]" except at the beginning of a sentence: "Reference [3] was the first ..."

Number footnotes separately in superscripts. Place the actual footnote at the bottom of the column in which it was cited. Do not put footnotes in the abstract or reference list. Use letters for table footnotes.

Unless there are six authors or more give all authors' names; do not use "et al.". Papers that have not been published, even if they have been submitted for publication, should be cited as "unpublished" [4]. Papers that have been accepted for publication should be cited as "in press" [5]. Capitalize only the first word in a paper title, except for proper nouns and element symbols.

For papers published in translation journals, please give the English citation first, followed by the original foreign-language citation [6].

REFERENCES

[1] G. Eason, B. Noble, and I. N. Sneddon, "On certain integrals of Lipschitz-Hankel type involving products of Bessel functions," Phil. Trans. Roy. Soc. London, vol. A247, pp. 529–551, April 1955.
[2] J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68–73.
[3] I. S. Jacobs and C. P. Bean, "Fine particles, thin films and exchange anisotropy," in Magnetism, vol. III, G. T. Rado and H. Suhl, Eds. New York: Academic, 1963, pp. 271–350.
[4] K. Elissa, "Title of paper if known," unpublished.
[5] R. Nicole, "Title of paper with only first word capitalized," J. Name Stand. Abbrev., in press.
[6] Y. Yorozu, M. Hirano, K. Oka, and Y. Tagawa, "Electron spectroscopy studies on magneto-optical media and plastic substrate interface," IEEE Transl. J. Magn. Japan, vol. 2, pp. 740–741, August 1987 [Digests 9th Annual Conf. Magnetics Japan, p. 301, 1982].
[7] M. Young, The Technical Writer's Handbook. Mill Valley, CA: University Science, 1989.

IEEE conference templates contain guidance text for composing and formatting conference papers. Please ensure that all template text is removed from your conference paper prior to submission to the conference. Failure to remove the template text from your paper may result in your paper not being published.