BOILERPLATE FOR TESTING DIFFERENT STYLES OF PEER-TO-PEER NETWORKING FOR MULTIPLAYER GAMING

Jamie Sullivan-Phillips & **Henry Finston-Perry** (CSC 466)
Department of Computer Science, University of Victoria

1. INTRODUCTION

The goal of this project is to provide an extensible, online game network simulator to demonstrate how different peer-to-peer (P2P) and client-server (C/S) network architectures can be applied in a video game context. The tool created for this project can simulate multiplayer game traffic and allow game developers to observe the performance of network architectures within the context of different game types. This will allow developers in the video game industry to make more informed decisions during the system design phase of production. A downloadable version of our simulator is included with this paper.[1]

While there has been research on methods for ensuring fairness of play and optimizing state management in a P2P network in online multiplayer game settings [1, 2], we have yet to find any research that specifically covers network architecture performance. Our strategy was to compare four different generalized network architectures, which are shown below in Figure 1.

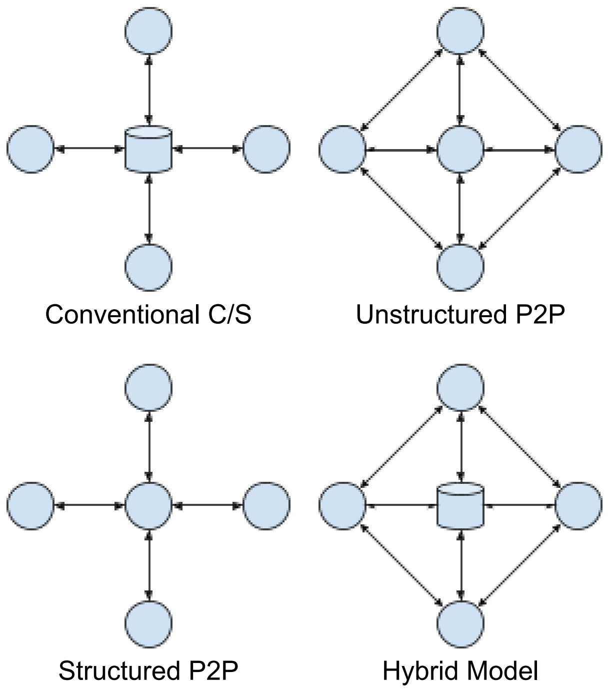

Figure 1. Representations of our minimal test cases. Circles represent a user node; cylinders represent commercial servers.

Many web applications—not limited to online games—have utilized each of these architectures to provide their respective services. Skype, for example, was a widely used video-telephony service initially based on an unstructured P2P architecture, before transitioning to a hybrid architecture and ultimately to a C/S model in order to provide reliability with increasing user demand [3]. Traditional C/S game architectures only require players to maintain a good connection to a central machine (or group of machines), reducing the bandwidth cost for each player; however, more and more game developers are choosing to rely on

[1] https://henryfinstonperry.github.io/csc466-termproject

P2P architectures as an alternative to investing in expensive infrastructure.

2. EXPERIMENT

The goal of our experiment was to run a simulation environment using a framework based on the C++ programming language called OMNeT++[2] and observe the round trip times (RTT) of packets in order to evaluate the performance of each network architecture. The goal was to construct a robust tool for developers to make more informed design decisions. The following subsections contain a summary of our reasoning for developing the simulator.

The first step in constructing a robust and representative simulator was researching the types of traffic that games send over a network in order to correctly emulate this traffic inside the simulator. Secondly, we needed to define key variables that would sufficiently distinguish several architectures. These variables are shown in Table 1, and we present our definitions for homogeneous and heterogeneous resources in Section 3.

	Homogeneous	**Heterogeneous**
Data rate	(80, 100) Mbps	(60, 120) Mbps
Latency	(5, 10) ms	(10, 100) ms
Drop rate	(1, 2) %	(1, 8) %
Input period	(10, 1000) ms	(10, 1000) ms

Table 1. Assigned values or ranges for networks with homogeneous and heterogeneous connections, which we found to be representative of real situations.

Using the OMNeT++ network simulator, we altered the parameters shown in Table 1 while the number of nodes stayed constant for each of the four architectures shown in Figure 1. The parameters, while somewhat arbitrary, are based on known averages for today's topology. This highly controlled simulation environment allowed us to compare the time for changes in the game state to propagate to all pertinent clients in a network, which effectively correlates to the delay between player input and a reaction in the game world. To aid in speculating on which architectures would be most appropriate for different types of games, the input period variable accounts for fast- and slow-paced games (i.e., action or turn-based).

To model the game state, we set and reset a single bit (boolean) value to represent a change in the world's state. For our P2P network implementations, there is a simple consensus mechanism that ensures each node will only accept updates to the game state if the packet's state counter is greater than its own. When a player inputs a change to the game world, that player negates its own world state and notifies the pertinent recipients (i.e., the server in a C/S model and neighbors in P2P models).

The first step to build the simulator was following a tutorial[3] to construct a simple "tic-toc" network consisting of two nodes that pass a single packet back and forth. Later sections of the tutorial introduce key features such as parameters, channels, and submodules. Our simulator expanded upon this simple tic-toc network with the use of the OMNeT++ Simulation Manual[4] to create microcosmic representations of C/S and P2P networks. Using modular parameters, we were able to create simulations to our exact specifications. While the data rate

[2] https://omnetpp.org

[3] https://docs.omnetpp.org/tutorials/tictoc/

[4] https://doc.omnetpp.org/omnetpp/manual/

and latency are built into OMNeT++, packet loss and distance were configured with custom C++ code.

3. DISCUSSION

In Table 1, we refer to “homogeneous” and “heterogeneous” scenarios. These terms are meant to differentiate a series of tests on nodes with similar network resources from those on nodes with vastly different resources. For instance, a homogenous scenario might place all players in the same city and with the same ISP; this, however, is rarely seen in practice as inter-ISP resources can vary just as much as user resources. Therefore, we have introduced heterogeneous scenarios where players might be on opposite sides of the world and have vastly different network speeds.

Our original plan was to use Wireshark[5] to observe the properties of network traffic while running a P2P video game to better understand the nature of video game packets. Next, the results of processes running on host peers were to be compared to those running as regular peers. We also intended to test network size to see how it affects performance.

Our goals evolved as we entered the development and testing phases of our project. Using our initial assumption that a P2P game designates one of the players to serve as the “host”, we wanted to measure the performance cost to the host node of maintaining the game’s current state and sending updates to other nodes. However, we quickly discovered that CPU/RAM usage is negligible due to the small packet sizes for games, and the fact that the majority of the game logic is performed on the client side rather than the host side. Only video and audio information is truly handled by the host node or server. With this discovery, we switched our strategy from simply performing tests with Python to focusing on creating a robust and verifiable simulation environment with OMNeT++.

Although Python and its many libraries offer more granularity compared to C++, we made the switch part way through the project to OMNeT++ for its proven stability and correctness. With Python, it was very difficult to verify that our built-from-scratch simulation was working correctly. One major bonus from switching to OMNeT++ was the visualizer. This feature makes our simulator more usable for the average person; the only requirement is installing OMNeT++. Unfortunately, OMNeT++ has compatibility issues with MacOS, which significantly slowed down our experiment process. This is an issue we had not run into with Python.

Lastly, while implementing packet loss and retransmission was out of scope for our particular experiment, we can hypothesize that P2P would suffer more from packet loss given there are more locations for failure to occur. We recommend this feature as well as several others in Section 5.

4. RESULTS

This section provides a summary of the experiments we conducted with the basic simulator. Figures 2 through 5 below show the RTT for a single node across architectures. For the full set of images in

[5] https://www.wireshark.org

high resolution, including graphs with all nodes overlaid, please see the included GitHub repository.

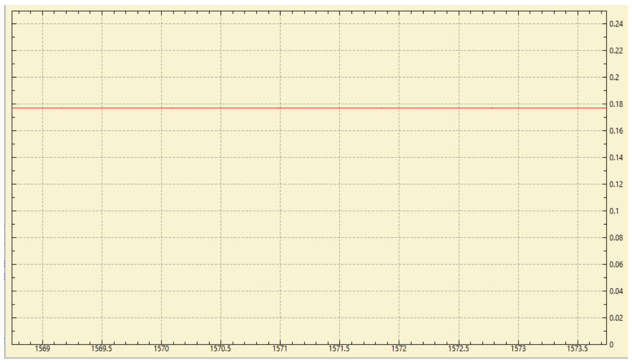

Figure 2. RTT in a local, homogenous client-server network is 0.17 µs.

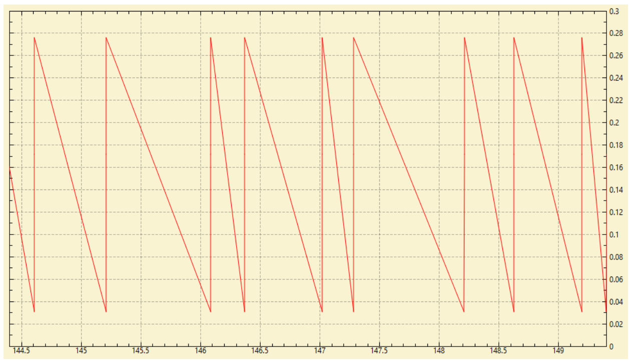

Figure 3. RTT in a local, homogenous unstructured P2P network is between 0.03 and 0.27 µs.

In Figure 3, a node in an unstructured P2P network is communicating over a set of links with a distance range of (2, 40) km and latency range of (5, 10) µs. Figure 2 shows the RTT of a single node in a C/S link with a distance of 25 km and latency of 5 µs.

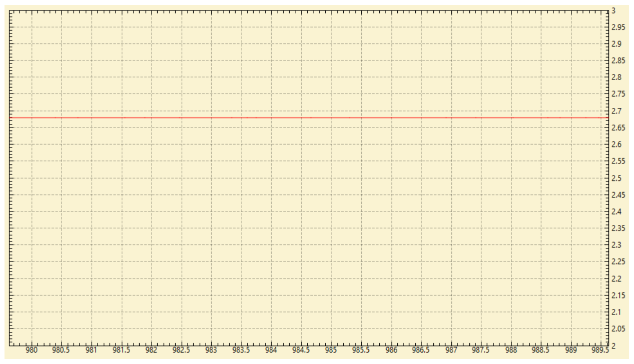

Figure 4. RTT in a regional, homogenous client-server network is 2.675 µs.

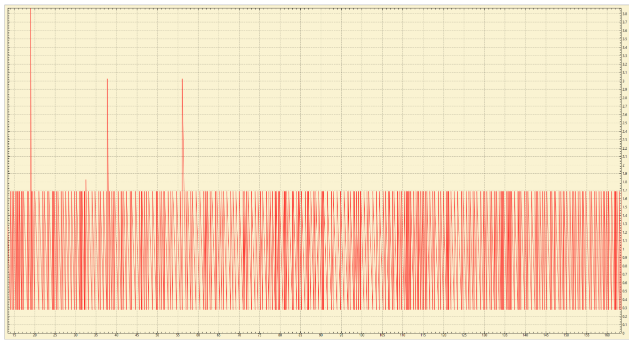

Figure 5. RTT in a regional, homogenous unstructured P2P network is between 0.29 and 1.69 µs with spikes of 3 to 4 µs.

Clearly, the values assigned can impact the results heavily, so our focus is on the consistency of RTTs rather than their magnitudes. High levels of inconsistency in time between player actions and world updates can negatively impact player experience in fast-paced games such as first-person shooters. In turn-based strategy games however, this inconsistency would not be a significant problem as there is a natural delay between action and reaction. Therefore, different game types should investigate the needs of their application and employ the network technique that best matches their games needs.

6. CONCLUSION

Since we have provided an extendible "boilerplate" simulator along with this paper on which we conducted our experiment, there are several areas in which we recommend further exploration. Firstly, one area we initially hoped to explore was incorporating network address translation (NAT) devices into the network, since NAT devices are widely used and impact P2P speeds [4]. This capability is possible in OMNeT++. Another recommendation we would offer is implementing network partitions between peers to observe its impact. Finally, while OMNeT++ includes a visualizer within the C++ application, it is possible to use Python[6] to analyze results and create more complex visualizations.

REFERENCES

[1] B. Knutsson, H. Lu, W. Xu & B. Hopkins. "Peer-to-Peer Support for Massively Multiplayer Games." IEEE INFOCOM 2004.

[2] S. Rieche, K. Wehrle, M. Fouquet, H. Niedermayer, L. Petrak & G. Carle. "Peer-to-Peer-Based Infrastructure Support for Massively Multiplayer Online Games." IEEE CCNC 2007.

[3] S. A. Baset & H. G. Schulzrinne. "An Analysis of the Skype Peer-to-Peer Internet Telephony Protocol." IEEE INFOCOM 2006.

[4] Y. Liu & J. Pan. "The impact of NAT on BitTorrent-like P2P systems." IEEE INFOCOM 2009.

[6] https://docs.omnetpp.org/tutorials/pandas/