Performance Evaluation Report

Overall, I am happy with the outcome of the project. I was able to implement most of the features I set out to and got a functional end result in a mostly coherent project. I learnt a fair bit along the way about ENet and general packet creation and sending, and Reliable vs Unreliable transmission. Though, my project will only ever be a toy/PoC level implementation. In the real world, you would likely use different technologies and techniques to achieve this result which I'll discuss further below.

Issues Encountered

One thing I didn't consider was that ENet is primarily a library used for implementing networked multiplayer in a game project. It works around creating a Host and having a finite amount of concurrent peers connect to that Host and exchange packets over a connection life-time. I utilise ENet for my non-gameplay related network communication, like directing a user to another server and authentication. In practice, this would likely be handled by something like HTTPS requests and not UDP / socket based traffic. It felt a bit weird to achieve my result using ENet, but given the academic nature of the project, ENet was fine for getting some C++ and basic networking reps in.

Another issue I ran into was the complexity of having a Windows C++ application start another process that was not tethered/child to the current process. I ultimately dropped support for this as it was not critical to the outcome or concept I was trying to achieve. I would like to revisit it in future.

Performance of the System

I am happy with the performance of the system. Memory footprint across the modules is low, as is processing time for some complex tasks.

One particular task that I thought would be slow is the loading and lookup of country codes based on IP address. It took some time to think how to store this data and I am happy with my result. Loading and building the IP Range database takes about 315ms on my NVMe SSD, and 850ms on my SATA SSD. Searching an IP address that exists in the database takes about 1100 nanoseconds.

Optimisations

The 'Ping all Balanced Servers' request is not ideal. It relies on connecting to a bunch of ENet sessions, sending some data and then just measuring the ping provided by ENet. Something lower level would be much more performant here, but also require me to implement my own networking stack.

Areas of Improvement

I used ENet for all networking in this project, but ENet is designed specifically for 'game lobbies' and only a small portion of my project is actually that – So utilising a more general networking stack like Winsock would likely be cleaner for this infrastructure.

Additionally, some flexibility in configuration like sending users directly from the Master Server to Game Sessions would be cool. As it stands, you would need to bunny hop through a Balanced Server with just one Game Session connected to it.

Required Changes

One thing I didn't consider was how the end user might hook into the behaviour of the various modules for their own use. E.g perhaps the end user wants to know when the client successfully connects to a master server or balanced server. Perhaps they want to connect to a Game Session manually rather than automatically. I ended up creating some public callbacks. Function pointers that the end user can subscribe to. These are available in all 4 modules for Connect, Packet Received and Disconnect so the implementer is able to do their own responses to various connection events.