CM649: Advanced Programming Issues for Games
Assignment 2009-10

Claire Blackshaw
s027525

January 25, 2010

1 Product

1.1 Network Model

The Java data stream class is very poorly suited to game data transmission or any high speed sending of predicatable data. The data needs to be serialised and then streamed to reciever. This can lead to partially delivered packages and a requirement for buffering.

The ideal solution is a struct which can be broken into packets and an underlying layer which buffers and tracks transmissions. The size is predictable, can be bit packed and sanitised.

A comprimise can be made by packing the data into character sized chunks which are transmitted in two byte pair. We know the order is mantained by the underlying TCP system. So we can throw away anything that is not divisible by two. A slight restructing of data compressed everything into two bytes.

The next problem is the server based calculations. This model cannot be cheated by the clients but is horrible unsuited to games.

The response time from player input to visual feedback is at minium double the current latency. Putting aside the horrible render model being used by the Java client this is the greatest improvement which could be made to the client. [1]

This could be solved by placing some trust in the client. If the position of the local bat is updated imeddiatly on the client, not waiting on the network loop.

The next upgrade requires sending time-stamped transforms. By time stamping transmission, and sending full transforms as opposed to merely a position then more accurate movement information can be used for calculation.

The ball could also be tracked and any motion extrapolapoted to make the ball update appear instant. Any network spikes will not be visually obvious.

[1] The changes are large and have not been made as it is out of the scope project but details are listed here to demonstrate understanding and address issues found in the implementation of the new features. Useful Source: `http://www.flipcode.com/archives/Network_Game_Programming-Issue_07_I_bent_my_Wookie.shtml`

This does require the client has an understanding of the collision model. Whenever a new transform is recieved from the network the local system can slowly bring the two models into sync avoiding visual snapping.

This drastically reduces the dealy in the feedback loop, which is now wholly constained by the renderer and input lag. Resulting in what seems to the player a much more responsive game. The slow redraw and server driven model leads to a series lag time.

1.2 New Feature: 2D Ball

2D bat movement was requested as a new feature. It requires a large amount of changes to both the server and the client.

As well as new data transmission formats. We are still using the simple point collision model, which in the worst case could result in the ball traveling through the bat.

A better method would be a ray-segment collision model which prevents this issue. It also opens more sophisticated collision options.

1.3 New Feature: Rounded Bat

The new feature is a rounded bat. We get the point at which the bat collides with the ball. We can then modify the X component of the ball direction based to how close to centre the collision was.

This change is purely server side as a minor alteration to the collision code. This gives the illusion of spin without requiring the complex upgrades previously discussed[2].

1.4 Java vs. C++ Server Structure

The comparison of Java vs C++ is best done on a case by case basis. Some generalisations can be made.

C++ has a higher recorded performance rate, but needs to be compiled for the specific machine and often needs minor changes based on the platform. C++ also offers a wider range of libaries and tools.

Java runs in a virtual machine allowing it to run on a variety of machines but it suffers on the performance side.

On as small scale such as this tennis project the performance issues are negligible. C++ has been proven to scale better than a Java based system, though other high level languages have performed well in this area as well.

The Java data stream is a horrible system for game data. It also reverses the data, which C++ server then needs to invert.

At a language level Java handles strings, localisations and encoding more efficiently. C++ has some excellent libraries written to handle strings, localisations and encoding. So while it seems like an advantage to Java the advantage is negated.

In this case certain features life the read-UTF8 are unusable as its almost impossible to format the data on the socket in fashion that the Java client will read correctly, even when the exact same byte stream is sent as an echo. This

[2]New ray-segment collision model and time stamped transforms.

can be worked around with some awareness. In a full blown system the sockets layer would be hidden behind a network objects and events system which would handle the interpretation of data.

1.5 TCP vs UDP

UDP uses datagram sockets to setup a minimal transport layer with the smallest overhead possible. TCP uses segments to have a transport layer with more systems.

TCP is Reliable, Ordered, Heavyweight, Streaming, able to track a connection and uses Segments. UDP is Unreliable, Not ordered, Lightweight, uses Datagrams.

The quick and easy way is TCP.[3] It provides a built in reliablity, sockets are easy enough to setup and built in acknowledgements. TCP does have a larger overhead to give these extra features, which does result in more lag.

UDP [4] suites fast unreliable trasmission. Perfect for voice communication or non-essitianl data. Such as decals, and stuff happening further off.

UDP is harder to sanitise and track the loss of connection or issues. The only checking on UDP is an integrity check. Any reliability or acknowledge layer needs to implemented by your code on top of the UDP framework. Too much safety, reliablity or checking might increase your overhead to the point at which TCP becomes a more attractive option.

So once your game needs the performance edge the move to UDP becomes logical. Almost all professional game engines use a UDP system with the reliablity sections moved higher up into the logical structure.

This allows the engine to have multiple streams or channels and flag data. Of course if any packet which contains data which needs to be ordered or reliable means the entire packet needs to be delievered in a reliable and ordered fashion. Though to be honest due to the acknowledgement and buffering system most engines use that if you want reliable packets you get ordered for free.

2 Documentation

2.1 Concurrency and Multi-threading

1. Write an introduction to concurrency and multi-threading, describe how it has been used in the game, and, referring to the development task, give a critical evaluation of its usefulness.

2.2 Basics of TCP/IP

The most common form of trasmission on the internet is TCP[5]. TCP lies on top of the IP[6] framework.

[3]Source: `http://gafferongames.com/networking-for-game-programmers/udp-vs-tcp/`

[4]User Datagram Protocol

[5]Transmission Control Protocol

[6]Internet Protocol

IP works by parcelling all data into packets. A packet has two parts a header and a body. The packet cannot exceed the MTU[7] which is typically 1500 bytes. So any information which is larger than that needs to be split into multiplte packets.

The header contains a good deal of information of the packet. Who sent it, who its intended for, and additional information. The header can be expanded by various things and added onto by various intermediataries. For instance if you past through a DHCP system, or routing the header can be altered.

The idea behind network packets and routing is that there are multiple paths from A to Z and they pass through all diffferent parts of the alphabet. Running a quick trace root function on your machine will give you an idea of the kind of paths which are used.

Each packet could take a different route, and because this is all based on assumption things can go horribly wrong. Network congestion, packet inspection, or unpredictable network behavior in general can result in packets going missing, arriving in a different order they were sent.

TCP was constructed to detect and manage a lot of these issues. Hopefully reducing traffic, and making the system easier to use. TCP places a layer of abstraction which deals with some underlying network issues but as a result will add some overhead in processing and increased header size. Meaning more packets will need to be sent and tracked.

The TCP format introduces a new data concept called a segment which is where the increased overhead. It introduces time-out and flags to manage the transmissions.

Unlike pure IP the TCP format introduces the concept of a socket. A socket is established between two machines and a port agreed. The socket then will either send or listen.

All transmissions are tracked, order correct and lost packets resent till the TCP segment is complete. A nice flow diagram is availible here on wikipedia [8].

2.3 Testing

The basic motto is, *When is doubt, print it out!*. Writing network code between two different languages debugging is extremly important.

Breakpoints can hardly ever be used as network traffic is lost and heartbeats[9] go silent. So printing out logs and analysis is a good system.

White box testing is the most common case and involves testing software which you understand and have access to the code. Black box testing revolves around inputs producing known outputs, without knowledge of the internals.

2.3.1 Unit Testing

Most network frameworks require a large set of unit tests. A unit test is a small function which can does a small set of tasks based on set of input, and compares them against known predictable outputs.

[7]Maxium Transmission Size

[8]`http://en.wikipedia.org/wiki/File:Tcp_state_diagram_fixed.svg`

[9]A system of ensuring the connection is still alive.

Unit testing while appears to be black box it operates at such a small scale that in reality it often needs some white box knowledge. Which muddies the definition somewhat.

These can run through a range of different data formats and check the transmissions.

Unit testing works quite well for testing small pieces of logic in a larger system. It allows you to track a large complex system, and if regular test runs are part devolpment schedule or nightly build then many bugs can be picked up as soon as possible.

This also narrows down the error to a portion of logic. This automated testing system requires little work to mantain and is highly effective.

2.3.2 Latency Testing

The most reliable extensive case study of games network overheads in recent history was performmed by Bungie using Halo 2[10].

The paper goes into some nice details but the modern consumer can be assumed to have the following.
Bandwidth: 8kb - 46kb
Latency (One-Way): 25 - 250 milliseconds
Packet Loss: 2% (Typical) - 10% (Worst Average)

As a network programmer this means the game should function at 8kb, with 250 milliseconds latency, and a packet loss of 10%. The game does not need to be perfect at this spec but it should hold together without dropping the connection or damaging other players.

Testing this using code inside the software is unreliable for two key reasons.

The first is changing the code for the purposes of testing will change the logic of the software. This can result in issues which would not otherwise occur and solve others which would otherwise occur. A few simple printf can break or solve timing related issues. Network issues are often timing related.

The second is the network stack. If the transmission is stopped in code then the network stack, and underlying systems are not used to their full potentional.

The solution to this problem is to setup a linux box with two network interfaces. The testing then needs to be done on a LAN [11] to be more controlled. The linux box can artifically delay packets, limit the throughput or just not send on some on.

This is a classic black box test as the linux box has no knowledge of the system and the testing can be performed with production builds, thought debug builds often provide more information when things go wrong.

This is a successful system for testing how well the game handles poor network conditions.

[10]Released originally and in more detail in a White Paper availible to registered devolpers (with slightly more detail) was republished at GDC 2008: Networking with the XNA Framework as part of the XNA documentation series. The GDC version can be found here `http://www.microsoft.com/downloads/details.aspx?FamilyId=E031C413-43F1-4C9E-92D2-7CDD64864D02&displaylang=en`

[11]Local Area Network

2.3.3 NAT Issues

Another Black Box test for network issue which often occurs, espcially when match making servers are involved are NAT[12] issues.

There are three basic forms of NAT: Strict, moderate, and open.

Strict to strict session negotiation can be tricky and often requires routing through an open or moderate NAT. The software needs to be aware of these issues.

The only way to reliably test this is multi-site testing. The different testers need to behind different NATs and with as a horrible a trace route as possible. Devolpers often resort to sending employees home and testing via home connections. Occasionaly QA will be outsourced to companies which have multiple offices and can perform these tests.

[12]Network Address Travesal