Asset and State Synchronization of Peer-to-Peer Multiplayer Online Games Over Named Data Network

Zening Qu
breezeway@ucla.edu / quzening@gmail.com

Problem

Massively Multiplayer Online Game (MMOG) players nowadays demand their games to have high playability degree while preserving game state consistency [1], which challenges network designers to provide robust game architectures and efficient synchronization mechanisms.

Game architectures can be roughly classified into three categories: client/server architectures, distributed architectures and peer-to-peer architectures [2]. Client/server architectures are the most commonly used in commercial products and distributed architectures are widely accepted by the academia as the best solution for MMOGs[2, 3], although it is peer-to-peer architectures that promise the best performance in terms of robustness and latency.

There are two major hurdles that are limiting peer-to-peer architectures' popularity. The first hurdle is that peer-to-peer games often need multicast connections between their peers to avoid network congestion and to make themselves scalable. Unfortunately IP multicast is not yet widely available[3]. The second hurdle is that peer-to-peer games often require more complex synchronization mechanisms than client/server architectures do. Other reasons why peer-to-peer games are not so popular include: (i) cheating is possible (ii) weak control for the game publisher [3].

If there is a way to make peer-to-peer games scalable and efficiently synchronized, game developers and consumers will benefit greatly from the augmented robustness and lowered latency brought to their games by the good architecture.

Approach to solving it in NDN

Named Data Network (NDN) can make peer-to-peer architectures scalable thanks to its two important features. First, the interest-driven nature of the network makes multicasting an easy task. Second, by caching content in network nodes, NDN effectively reduces network congestion. These two features will remove the restriction of bandwidth requirements and thus scaling up peer-to-peer games.

Besides scaling the game, NDN also offers a useful synchronization mechanism. For game asset changes, such as player joined or object destroyed, which happens in an order of seconds or even minutes, CCNx Synchronization Protocol (CCNx Sync) [4] gives a good example of a synchronization mechanism that is both scalable and loosely coupled with other modules in a game developer's code. For the more frequent game state changes, such as changes of a moving object's position, both the freedom and the burden of synchronizing the state is fallen to game developers. Usually these state changes are more application-specific than asset changes are.

Lastly, we also hope to provide game authentication and cheating avoidance methods by utilizing NDN's verification mechanism [5].

Implementation

To prove ideas we converted an open source, single player car racing game [6] provided by Unity 3d into a peer-to-peer MMOG where both asset and state changes were synchronized. Different game events were classified into different categories (asset change or state change) and implemented differently. Asset changes were written to a local CCNx repository [7] and advised through CCNx Sync [4]. State changes were synchronized by normal CCNx interest and content together with optimistic prediction and error correction algorithms, without going through the local CCNx repository.

The architecture of the car racing game is illustrated in Figure 1. Each peer runs a Unity application which renders the scene. The Unity application relies on ccnd to communicate (and synchronize) with other peers. To synchronize assets ccnr must also be running. Note that in Figure 1 each peer has a direct connection to all the other peers in the game. This will be discussed in the next section.

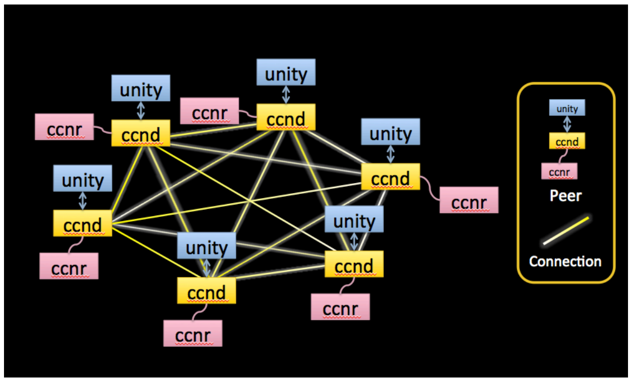

Figure 1 A fully-connected peer-to-peer NDN network with six peers

We were using CCNx C library to synchronize both assets and states. The rest of the game code is written in C#. Interoping between C# and C relies largely on the System.Runtime.InteropServices Namespace [8] and Unity Plugin [9, 10].

Discussion

As mentioned in the previous section, currently our peer-to-peer game still requires each player to be directly connected to all the other players, which is not truly scalable and does not take good advantage of NDN's web cache. Our problem is that the CCNx Sync protocol that we used for asset synchronization only operates at scope 2 (directly connected peers) for now. As soon as that restriction is removed, our peer-to-peer game will be truly scalable. Also note that the network congestion problem caused by peer-to-peer architectures is no longer a hurdle.

Future work

1. To make our peer-to-peer game truly scalable we will need an asset synchronization mechanism different from our current one. We may have to search for or design a new synchronization protocol that can work for peer-to-peer networks at a scope equal or greater than two. We may need to narrow down to focus on synchronization protocols specifically designed for MMOGs.
2. Design methods that use NDN signature generation and verification mechanisms to avoid certain types of cheating or perform authentications in peer-to-peer networks. This work can be very program-specific and takes in-depth understanding of cheating patterns and authentication process.
3. State synchronization algorithms come in various categories. Prediction and error correction is suitable for car racing games but there are many other algorithms such as bucket synchronization, time warp and roll back [2] that may fit better into other games (for example first person shooting games). As network designers we need to study these available algorithms to learn how to provide better support for different types of games.
4. In order to maintain consistency among peers a synchronization mechanism must also know how to resolve conflictions. This problem is not yet addressed in our current game.

References

1. Cacciaguerra, S., et al., *Car racing through the streets of the web: a high-speed 3D game over a fast synchronization service*, in *Special interest tracks and posters of the 14th international conference on World Wide Web*2005, ACM: Chiba, Japan. p. 884-885.
2. Ferretti, S., *Interactivity maintenance for event synchronization in massive multiplayer online games*, March 2005, University of Bologna (Italy).
3. Cronin, E., et al., *An Efficient Synchronization Mechanism for Mirrored Game Architectures.* Multimedia Tools and Applications, 2004. **23**(1): p. 7-30.
4. *CCNx Synchronization Protocol*. 2012 2012-04-15 19:58:57 PDT [cited 2012 May 18]; Available from: http://www.ccnx.org/releases/latest/doc/technical/SynchronizationProtocol.html.
5. *CCNx Signature Generation and Verification*. 2012 2012-04-15 19:49:02 PDT [cited 2012 May 18]; Available from: http://www.ccnx.org/releases/latest/doc/technical/SignatureGeneration.html.
6. *Car Tutorial*. 2012 [cited 2012 May 18]; Available from: http://unity3d.com/support/resources/tutorials/car-tutorial.
7. *CCNx Repository Protocols*. 2012 Last updated 2012-04-15 19:58:57 PDT [cited 2012 May 18]; Available from: http://www.ccnx.org/releases/latest/doc/technical/RepoProtocol.html.
8. *MSDN Library*. Available from: http://msdn.microsoft.com/en-us/library/system.runtime.interopservices.aspx.
9. *Unity Manual - Plugins*. Available from: http://unity3d.com/support/documentation/Manual/Plugins.html.
10. Qu, Z., *Building CCNx bundle plugin for Unity 3d Pro on Mac OS*, in *Google Docs*2012.