NDN MOG Project Snapshot

Zening Qu

January 13, 2013

Contents

List of Figures

1 Introduction: Objective, Inspiration and Potential Findings

This document is aimed for the Named Data Networking Multiplayer Online Game (NDN MOG) research group. It is written at a time when a small scale car racing game was implemented and synchronized on the NDN testbed [12]. The author tries to provide a rich background of MOG design, in the hope of fostering the design and evaluation of a more extensive Role Playing Game (RPG) game over NDN. The author does not attempt to bring up too much details about NDN. Interested readers should turn to the project website (http://www.named-data.net/) and its publication list.

In this section we define the project objective and potential findings. Section 2 introduces the general game design of our planned RPG game and compares it with characteristics of World of Warcraft (WoW), the representative MOG. The MOG synchronization problem and related design issues are described in section 3. Finally, section 4 tries to solve the synchronization problem for our RPG game. Note that NDN MOG is an on-going project, hence section 4 is merely a best-effort solution for the time being. The document may raise more questions than it answers. Its content (including background and design) is subject to substantial improvement and change.

1.1 Project Objective: Explore The Game Synchronization Problem And The Related Problems

The primary objective is to explore the game synchronization problem on NDN. Closely related problems such as architecture choice, interactivity and scalability are seriously considered. These problems are depicted in section 3.

1.2 Potential Findings: Impact of NDN On P2P MOGs

Potential findings of the project include but are not limited to:

- A prototype RPG game on NDN, with synchronized game state
- An evaluation of the design of the game, regarding consistency, latency, scalability etc.
- A library/toolkit that supports game sync and general MOG development

2 General Game Design: LPP and WoW

In this section we present general game design of our prototype RPG game *LPP* (inspired by *Le Petit Prince*) and compare it with the representative MMOG World of Warcraft (WoW). We do not intend to create a game of similar complexity in terms of gameplay, but we hope to simulate interactions that are typical in WoW and find out the game's scalability. In this way, our experiment will carry more practical value.

2.1 Genre: Role Playing Game

Role-playing games (RPG) is the genre we want to study. Most existing MMOGs, like WoW, are RPGs [8]. These games allow thousands of concurrent players to interact in a shared game world, which poses a big challenge in network bandwidth. Although other game genres such as real-time strategy (RTS) games and first-person shooter (FPS) games also supports massive concurrent users, these games are often divided into small isolated sessions where a small group of players communicate exclusively within the group [8]. Therefore RTS games and FPS games present smaller challenges to the network and are not studied in this project.

2.2 Scale: 1 Million Concurrent Players

WoW is the world's most subscribed MMORPG. It has more than 10 million subscribers in worldwide [11, 13]. In 2008 its number of concurrent players was reported to reach a peak value of 1 million [9].

We are interested in the scalability problem because it is closely related to the synchronization problem (see 3). While maintaining game state consistency, we hope to scale our game up. The scale of WoW should act as a reminder of MMOG requirements from the industry.

2.3 Activities: Raiding and PvP

In most MMOGs, the base map, or the land, is immutable. Players experience the virtual world through sensibilities of their avatars. In [15, 14], the authors performed an action-specific analysis of WoW's network traffic and proposed the following taxonomy of game activities:

- Questing: avatars interact with non-player characters (NPCs) in the virtual world.
- Dungeons: a small group of five avatars work together to kill a series of more powerful NPCs; each group is isolated from other groups and players; replications of dungeons are instantiated for each group.
- Raiding: the same as Dungeons, except that NPCs are more powerful and group size is larger (10, 25 or 40).
- Trading: trade of goods between players or between a player and an NPC.
- PvP: player versus player combat; number of participants can reach 80.

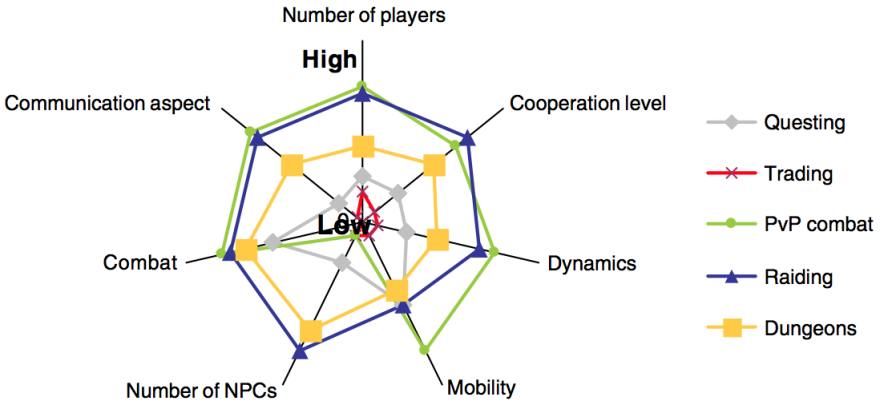
Figure 1 reports characteristics of different types of activities. Among the five action types Raiding and PvP combat involves the largest number of players, requires most cooperation and communication (figure 1a). Consequently, these two types have the highest requirement in bandwidth and latency (figure 1b and 1c). Because Raiding and PvP combat are so challenging, they deserve serious consideration in our experiment. We emphasis these two forms of activities in the design of our prototype game (see section 2.4).

2.4 LPP Gameplay

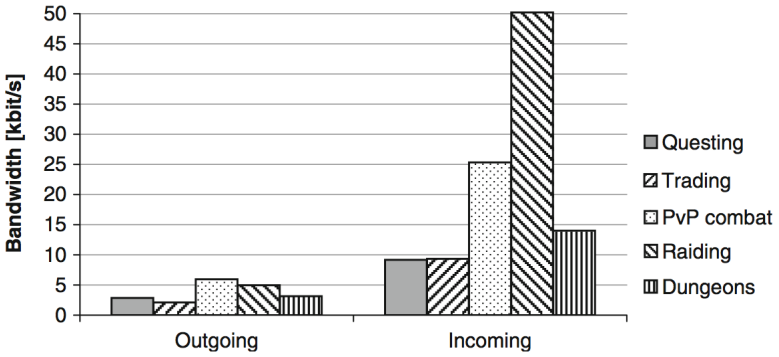
Our prototype RPG is inspired by Antoine de Saint-Exupéry's novel *Le Petit Prince*, which is how it got the name LPP. The novel is about a young and innocent alien prince who left his home planet for a long journey. He traveled to several asteroids and planets and witnessed many people and things that he considers absurd or "adult-like". During his stay on the earth, he met a fox who told him:"It is the time you have wasted for your rose that makes your rose so important." He tamed the fox, and the fox became unique and important to him. His experience and wisdom grew on this journey.

As mentioned in section 2.3, we are primarily interested in player actions similar to Raiding and PvP combat in WoW. Hence the LPP gameplay lies follows:

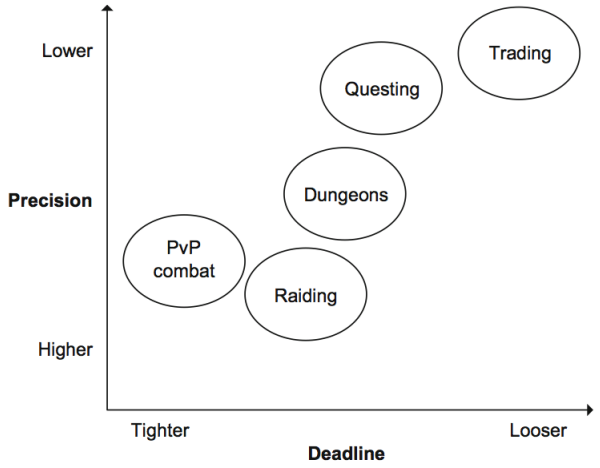
- Players take up the role of the prince.
- The game dynamically instantiates foxes, which are NPCs.
- Players try to tame foxes using spells. This corresponds to Questing in WoW.
- Players can form large groups to tame foxes. This corresponds to Raiding in WoW.
- Players combat with each other to practice their skills in casting a spell. This corresponds to PvP combat in WoW.
- As players tame foxes and practices with each other, their wisdom, or experience, grow.



(a) An overall comparison of various action types. Note that Raiding and PvP combat have higher requirements in multiple dimensions, which may lead to greater challenges to the network.



(b) Bandwidth requirement of action types. The label "Outgoing" means traffic from the client to the server. The label "Incoming" means traffic from the server to the client. Traffic is measured at the client's side. Node that Raiding and PvP combat consume much more bandwidth than the other action types.



(c) Latency requirement of action types. Raiding and PvP combat have the highest requirement in both precision and latency. This means that players actions should be more precise in terms of spatial accuracy when involved in these two activities, and they are given less time to perform these actions. In these scenarios, network latency is an especially sensitive issue.

Figure 1: Player actions evaluated under various metrics. All three graphs excerpted from [14]

3 The Synchronization Problem and Related Problems

This section tries to shed light on the MOG synchronization problem and its related problems by providing information from the literature. We summarized that the goal of the synchronization problem is to reach game state consistency (section 3.1) and explained why architecture choice, interactivity and scalability are closely related to the synchronization problem (section 3.3).

3.1 Consistency Models: Event Based (P2P) and Update Based (C/S)

Thousands of players experience a shared virtual world in a MMOG. Media of the virtual world (such as 3D models of the land, buildings, NPCs and characters) and the *game logic* (rules of how this world should function and how its members should interact) are typically installed on the player's machine. In addition to that every player needs a copy of the *game state* in order to learn the evolvement of the world. The game state is the state of the virtual world plus the state of all participating players, NPCs and objects. It is obvious that the game state changes as time passes by. Players interact with the world by generating actions that changes the game state. Because each player has a replication of the game state, a consistency issue arises: copies of game state should always be consistent with each other or else players' opinions of the virtual world would diverge. Researchers have proposed various models and algorithms to ensure such consistency.

Two consistency models exists: event-based and update-based (see figure 2) [7]. The event-based consistency model is used on fully distributed P2P networks where every peer is directly connected to *all other peers*. Each peer maintains a replication of the global game state and send *event*s that represent the player's action in the virtual world to all other peers. As newer global game state can only be computed when the game logic has sufficient input, i.e. actions of all relevant peers within a relevant time period, each peer actively listens to events published by all other peers. Moreover, due to the uncertainty of transmission delay events may arrive at a peer out of their generation order. Peers need a synchronization algorithm to figure out the generation order of received events in order to compute the correct game state. Common synchronization algorithms are time warp algorithm [5], trailing state algorithm [1], time bucket algorithm [6, 2] and their variations. These algorithms are discussed in section 3.2. Note that a peer could also use a reliable, ordered delivery service to make sure it receives all events and in the right order. Yet it is reported that such services would significantly degrade game performance [3], and are rarely adopted.

The update-based consistency model is used on C/S MOGs. Clients do not inter-connect with each other but communicate through a central server. Clients also do not have the right to compute the global game state – this is withhold by the server. As illustrated in figure 2b, clients merely generate events according to player actions and wait for *update*s of the global game state computed by the server. Clients then applie the update to its local game state. Depending on the lower layer services used by the client and server, a client may need to occasionally reorder its received updates. The central server also need a synchronization algorithm to process all received events in their generation order. The synchronization algorithms here are identical to those in the event-based sync model.

3.2 Synchronization Algorithms: Conservative and Optimistic

As previously mentioned, both event-based and update-based consistency models need synchronization algorithms (see section 3.1). In other words, both P2P MOGs and C/S MOGs need synchronization algorithms.

The game synchronization problem can be described as follows. Machine A is running a game sync algorithm. It is connected to some other machines. These machines share the same set of data. The data set does not grow but its value changes in a fast pace. All machines that are connected to A may request to update the data set but only A has the authority perform the update. So A has to gather all requests and know their relative order, otherwise the value of the data set would be wrong. In a C/S architecture, this machine A would be the server (figure 3b). It would gather client events and publish game updates. In a P2P architecture, this machine A could be any peer (figure 3a), as every peer holds the right to update a certain subset of the global game state. In a hybrid architecture (see section 3.3.1), this machine A would

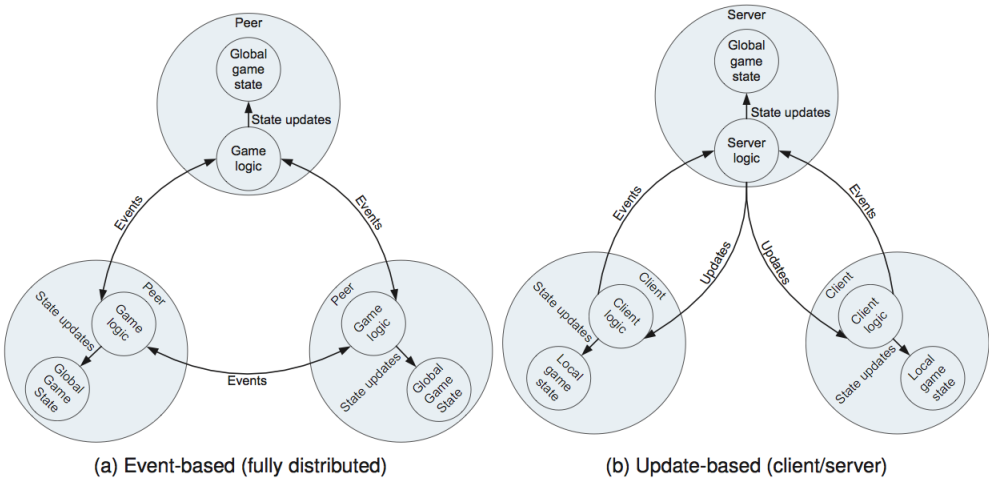
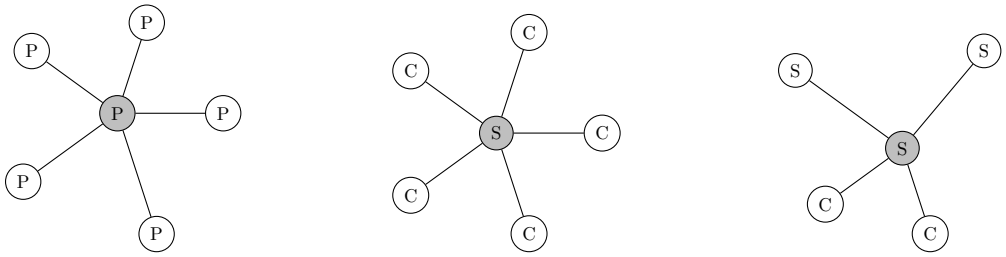


Figure 2: Consistency Models. Excerpted from [7]



(a) P2P (b) C/S (c) Hybrid

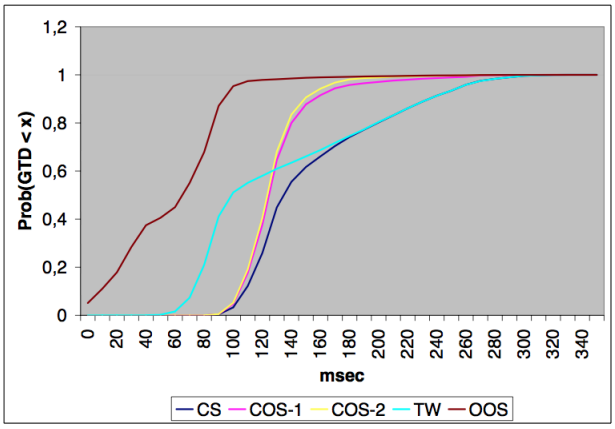
Figure 3: Synchronization in different architectures. C stands for client. S stands for server. P stands for peer. The shaded node represents the node that is running a game synchronization algorithm.

be one of the cluster servers (figure 3c). It would gather requests from both its clients and its peer cluster servers.

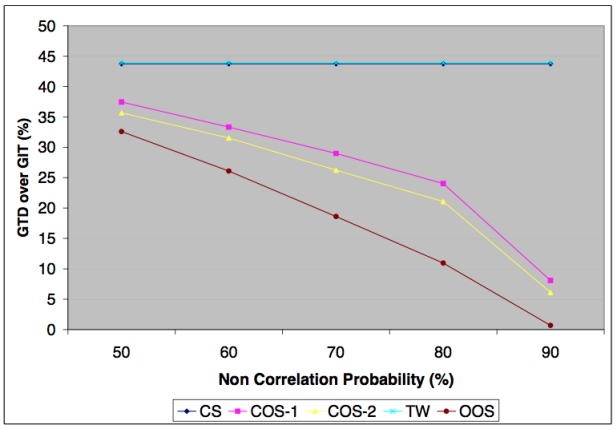
Note that game synchronization problems differs from sync algorithms for other applications (such as file syncing) in that (1) the content to be synced does not grow in size, but changes its value instead (2) the time constraint is much tighter. Such difference is reflected in the design of game sync algorithms, which emphasis on packet ordering and network latency.

Both conservative and optimistic synchronization algorithms have been proposed. Conservative algorithms are based on lockstep. Arrived events are delayed until it is safe to process them. Optimistic algorithms process arrived events once they are received, but performs game state rollbacks once it receives an event that arrives late. A simulation-based evaluation of these algorithms is shown in figure 4, from which we can see that optimistic sync algorithms outperform conservative ones (figure 4a).

In [4] the author proposed an algorithm to further improve optimistic sync performance by exploiting game semantics. The author exploited the "correlation" among received events and effectively reduced rollback ratios, thus enhancing performance (figure 4b). Similar strategy should be used in the LPP project (see section 4).



(a) Cumulative probability of packets' arrival. The vertical axis is the cumulative probability. The horizontal axis is the elapsed time. The quicker an algorithm reaches 1, the smaller latency it has. The tolerate threshold is 150ms according to the author of the graph.



(b) Percentage of packets with intolerable latency (more than 150ms). The vertical axis is the percentage. The horizontal axis is the "non correlation probability" of successive events. Large values would mean that the received events are less correlated, which would incur less rollbacks and consequently faster processing (note that the latency here includes the time spent in rollbacks).

Figure 4: Latency of various synchronization algorithms. CS stands for conservative synchronization. TW stands for time warp, which is representative of optimistic sync algorithms. OOS is a variation of time warp which was proposed by the author of [4]. COS-1, COS-2 are CS variations. Both graphs are excerpted from the same paper.

3.3 Related Problems: Architecture, Interactivity, Scalability, Availability, Security

3.3.1 Architecture Choice: P2P or Client/SeverClusters

Architecture choice is closely related to the sync problem. MMOGs are typically either P2P or Client/ServerClusters. Even though synchronization algorithms apply to both architectures, the performance might differ. Because servers aggregate client events, in C/S architectures the sync task involves machines but each would have a higher packet rate. On the contrary, in P2P architectures, there are more peers to listen to but they would have lower packet rates. Both the number of participating machines and the packet rate affects sync performance, and the author does not know yet which one is more significant.

3.3.2 Interactivity: MMORPGs are Latency-Sensitive

Interactivity is often used to evaluate the performance of a sync algorithm. Interactivity is measured by the latency between a game event's generation time and process time. This is comprised of two parts: the network transmission delay and the synchronization delay (locksteps, rollbacks etc.) []. The smaller the delay is, the more interactive MMOGs will be. According to literature, the tolerable latency for RPGs is usually between 100ms and 200ms [].

The author believes that P2P MOGs enjoy smaller latencies than C/S MOGs do. This is because peer are directly connected with each other and does not need to communicate through the server. However, the author haven't yet found sufficient data that supports this claim.

3.3.3 Scalability: P2P Is Scalable, But Not Its Consistency Model

Scalability is related to the consistency models (section 3.1) and the architectural choice (section 3.3.1). P2P networks are generally believed to be more scalable as they don't have centralized servers which are often bottlenecks. However, with a consistency model in figure 2a, the scalability quite limited, as each peer needs to be connected with all other peers.

There is a trade-off between scalability and latency. As mentioned in section 3.3.2, fully connected topologies bring smaller latency. But they also degrade scalability. In order to scale P2P games up, *overlay* networks have been used in P2P MOG simulations. For example, in [8] the authors used P2P overlay networks and Application Level Multicast (ALM) to scale their P2P MOG up to 4,000 players. When overlay network is used, peers no longer have to be fully connected with each other. The overlay network provides efficient routing for the peers. In Pastry [10] for example, messages are expected to be routed to their destinations within $log_{2^b} N$ routing steps, where N is the number of peers and b is a configuration parameter. When $b = 4$, in a network of 1 million nodes, a message is expected to be routed to its destination within 5 hops. Figure 5 illustrates that most multicasted messages were delivered within 2 hops in the 1000 and 4000 players simulation of Knutsson et al. This concludes our example where latency is traded for scalability. The author believes that similar trade-off is also doable in NDN.

It is also worth mentioning that in the simulations in [8] ALM infrastructure was used to provide application level multicasting. The authors of the paper reported that ALM degrades game performance. Since NDN is intrinsically multicasting, no ALM will be needed. This is a potential area where NDN could bring performance increase.

Finally, researchers agree that RPGs should exploit players' *locality of interest* to increase their scalability. Players form interest groups. Group members receive the same events. In [8] grouping is based on regions (figure 6).

4 Designing The Next Game: LPP

This section presents the technical design of our next prototype: the LPP game. General design considerations of the game such as genre, gameplay etc. can be found at section 2.

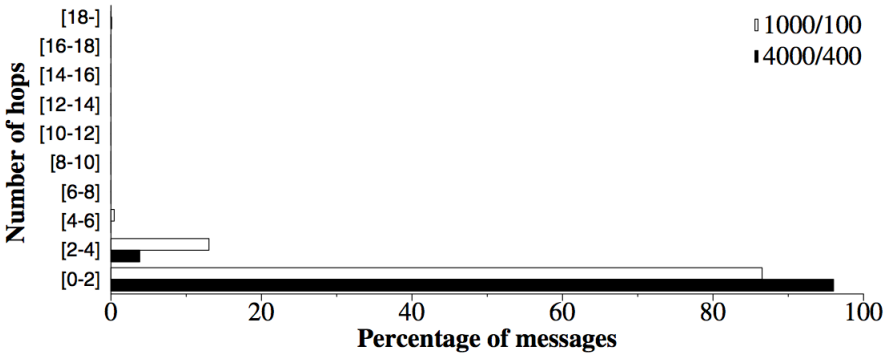


Figure 5: Distributions of multicast message hops, using Pastry. 1000/100 denotes 1000 peers in 100 groups. 4000/400 denotes 4000 peers in 400 groups. Figure excerpted from [8]

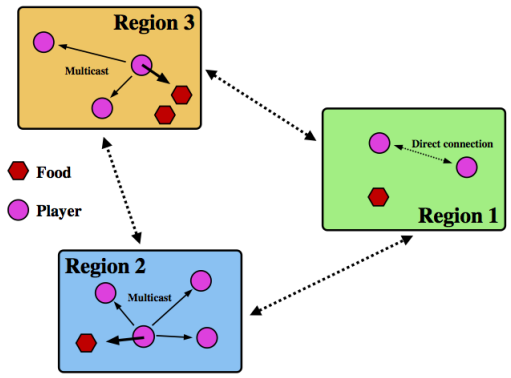


Figure 6: Locality of interest. Excerpted from [8].

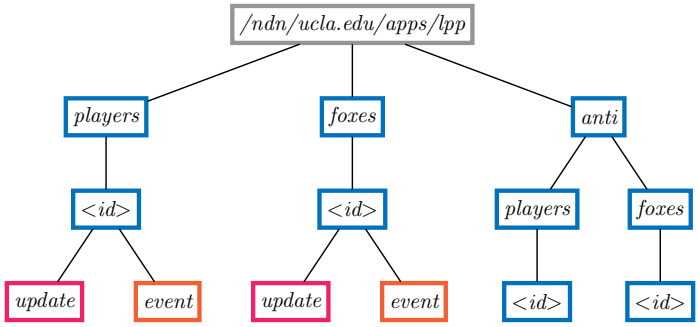


Figure 7: The LPP namespace.

Table 1: Content of named objects

named object	values
.../players/<id>/update	position, experience
.../players/<id>/event	tame
.../foxes/<id>/update	position, health
.../foxes/<id>/event	

4.1 Namespace

Namespace design of the LPP project is illustrated by figure 7. All content objects are to be synchronized under the `/ndn/ucla.edu/apps/lpp` namespace. Each player will be assigned a universally unique id, and will be able to generate both `update` and `event`. The name `update` represents game state of the corresponding player, such as position and experience. The name `event` represents the action of the player. For now the only possible action is "tame". Players will use this action to tame foxes and combat with each other. Names for NPCs (foxes) are similar to that of players. The `anti` namespace is explained in section 4.2.

One question is where game state data should be hosted. In C/S architectures, all player and NPC information should be hosted by servers and only servers can update the information (by publishing `.../players/<id>/update` and `.../foxes/<id>/update` objects). Clients will generate `event` objects and wait for the result state change. In P2P architectures, player information is hosted on the player's machine, and the player holds the authority to publish his/her own `update`. Players send `event` to each other and wait for the result state change in other players. Foxes in P2P games must be assigned *coordinators*, who are active players in the game. The coordinator's machine hosts a fox's state and updates it periodically.

We propose to use different strategies to deliver `update` and `event`. Updates are delivered in an unreliable yet ordered way, so no synchronization algorithm is needed in asset dissemination (see section 4.3). Events, however, need to be transmitted both reliably and in order, so we will use an optimistic sync algorithm to guarantee its order (see section 4.4).

4.2 Membership Service and Object Discovery: CCNx Sync

Players in the same area of interest (AoI) need to know each other's existence and all NPCs' existence. In our previous prototype game this was done using CCNx Sync. For P2P games, CCNx Sync does not require fully connected topologies, which makes the network more scalable. However, network latency may greatly increase when the network scales up. This is because there is no guarantee how long it will take for a message to reach its destination. On contrary, in a P2P overlay network the expected number of hops is within $logN$ (see section 3.3.3). The author believes that for a P2P MOG to be scalable, a routing mechanism similar to

that of overlay networks is necessary.

By the time this document is written, CCNx Sync still persists in the namespace design of LPP, though it is likely to be replaced due to the above mentioned reason. In figure 7, all blue-bordered nodes are associated with CCNx Sync. The `anti` namespace is designed to represent the deletion of a node. For example, when a player exits, there will be an `.../anti/players/<id>` node.

4.3 Unreliable, Ordered Delivery of Updates

Updates should be delivered in an unreliable, ordered fashion. Update publisher generate new objects periodically and frequently. Update subscriber send NDN Interest packets for the newest updates by setting the *RightmostChild* selector and using the exclusion filter. In this way, the timestamp of received update data packets will be in a non-decreasing order. So no synchronization algorithm is needed for updates. Note that using such delivery is very likely to cause packet loss. But we reason that such loss is tolerable because (1) updates are generated at a high pace (2) losing updates does not hurt game state consistency (3) the lost of packets can be compensated by other mechanisms such as *dead reckoning*.

4.4 Reliable, Ordered Delivery of Events

Events are to be synchronized reliably and in order because events will lead to state change, and losing events or confusing their order will lead to inconsistency. Optimistic synchronization algorithms will be used for event synchronization. Performance of candidate synchronization algorithms is presented in figure 4.

Events will be published as Interest packets. The replying Data packet should contain the result state update.

References

[1] Eric Cronin, AnthonyR. Kurc, Burton Filstrup, and Sugih Jamin. An efficient synchronization mechanism for mirrored game architectures. *Multimedia Tools and Applications*, 23:7–30, 2004.

[2] C. Diot and L. Gautier. A distributed architecture for multiplayer interactive applications on the internet. *Network, IEEE*, 13(4):6–15, July/August 1999.

[3] S. Ferretti. *Interactivity maintenance for event synchronization in massive multiplayer online games*. PhD thesis, University of Bologna, 2005.

[4] Stefano Ferretti and Marco Roccetti. Fast delivery of game events with an optimistic synchronization mechanism in massive multiplayer online games. In *Proceedings of the 2005 ACM SIGCHI International Conference on Advances in computer entertainment technology*, ACE '05, pages 405–412, New York, NY, USA, 2005. ACM.

[5] R.M. Fujimoto. Parallel and distributed simulation. In *Simulation Conference Proceedings, 1999 Winter*, volume 1, pages 122–131, 1999.

[6] L. Gautier and C. Diot. Design and evaluation of mimaze a multi-player game on the internet. In *Multimedia Computing and Systems, 1998. Proceedings. IEEE International Conference on*, pages 233–236, June/July 1998.

[7] J.S. Gilmore and H.A. Engelbrecht. A survey of state persistency in peer-to-peer massively multiplayer online games. *Parallel and Distributed Systems, IEEE Transactions on*, 23(5):818–834, May 2012.

[8] B. Knutsson, Honghui Lu, Wei Xu, and B. Hopkins. Peer-to-peer support for massively multiplayer games. In *INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*, volume 1, pages xxxv+2866, March 2004.

[9] The9 Limited. Blizzard entertainment's world of warcraft: The burning crusade surpasses one million peak concurrent player milestone in mainland china. http://www.the9.com/en/news/2008/news_080411.html, April 2008.

[10] Keong Lua, J. Crowcroft, M. Pias, R. Sharma, and S. Lim. A survey and comparison of peer-to-peer overlay network schemes. *Communications Surveys & Tutorials, IEEE*, pages 72–93, March 2004.

[11] Acquire Media and Blizzard Entertainment Inc. Alliance and horde armies grow with launch of mists of pandaria. http://www.vivendi.com/wp-content/uploads/2012/10/121004_ATVI_Alliance-and-Horde-Armies-Grow-with-Launch-of-Mists-of-Pandaria1.pdf, October 2012.

[12] Zening Qu and Jeff Burke. Egal car: A peer-to-peer car racing game synchronized over named data networking. Technical report, UCLA, October 2012.

[13] Diane J. Schiano, Bonnie Nardi, Thomas Debeauvais, Nicolas Ducheneaut, and Nicholas Yee. A new look at world of warcraft's social landscape. In *Proceedings of the 6th International Conference on Foundations of Digital Games*, FDG '11, pages 174–179, New York, NY, USA, 2011. ACM.

[14] Mirko Suznjevic, Ognjen Dobrijevic, and Maja Matijasevic. Mmorpg player actions: Network performance, session patterns and latency requirements analysis. *Multimedia Tools and Applications*, 45:191–214, 2009.

[15] Mirko Suznjevic, Maja Matijasevic, and Ognjen Dobrijevic. Action specific massive multiplayer online role playing games traffic analysis: case study of world of warcraft. In *Proceedings of the 7th ACM SIGCOMM Workshop on Network and System Support for Games*, NetGames '08, pages 106–107, New York, NY, USA, 2008. ACM.