Latency & State Consistency in Networked Real-Time Games

Christopher D. Canfield
cdcanfie@bu.edu
BU MET CS 535: Computer Networks

1 Introduction

In networked real-time games, and indeed any distributed virtual environment, there is an inherent tradeoff between latency reduction techniques and the consistency of the distributed state [12], [14], [20]. By studying the impacts on users of network latency and distributed state inconsistency, developers, companies and other stakeholders in the networked real-time games industry can make informed decisions on which tradeoffs are appropriate in which cases. For example, introducing lag locally to help maintain state consistency may be a reasonable trade-off for a slower-paced massively multiplayer online role-playing game (MMORPG), but it is unlikely to be so for a game in the first-person shooter genre (FPS).

This paper is organized into two parts. The first part, consisting of sections 3 and 4, begins with a discussion of the impacts on users of network latency. It then describes eight methods for managing distributed shared state in network games. Though real-time digital games are the focus of this paper, many of the methods described are in fact useful in other scenarios, such as distributed military simulations, educational simulators, and other distributed virtual environment applications.

The second part, consisting of section 4, describes *Triangle Wars*, a networked real-time game that can simulate network latency delays. This application was used to test player perceptions of latency.

2 Impacts on User Satisfaction & Performance from Network Latency & State (In)Consistency

A number of studies have been performed regarding the tolerance of users on network delay and real-time games. The studies reported the following results:

- Delays under 50 milliseconds do not impact player performance [16]
- Delays over 50 milliseconds but under 100 milliseconds begin to have a slight impact on player performance, but are rarely noticed by the player [4], [9], [12], [16], [19]
- Delays greater than 100 milliseconds are noticeable in fast-paced real-time games with a first person perspective, such as certain types of racing games and games in the first-person shooter genre. Player performance begins to fall at this level. [1], [2], [16]
- Players will migrate away from FPS servers that result in a latency greater than 150-180 milliseconds [1]
- Player latency tolerances are higher when using third-person (such as over-the-shoulder) and top-down perspectives, and for slower paced real-time games, such as those in the real-time strategy (RTS) genre and massively multiplayer online role-playing game genre [10], [19]. For example, Nichols and Claypool found that client performance did not decline by much in *Madden NFL Football,* in which the player controls multiple players at a given time from a top-down perspective, until latency reached 500 milliseconds [15].

In a networked rock-paper-scissors experiment using real-time video and audio, Hashimoto and Ishibashi found that a user's mean opinion score, a subjective rating on the impairment of the game due to network latency, declined from 4 ("Perceptible, but not annoying") at 60 milliseconds of delay to approximately 2 ("Annoying") at 120 milliseconds [9].

Interestingly, Chen, Huang and Lei found that absolute delay due to the network mattered less to players than delay jitter and packet loss [3]. They found that "if players quit because they are frustrated by unfavorable network conditions, on the average, 10 percent of their dissatisfaction is caused by network latency, 20 percent by network delay jitter, 40 percent by client packet loss, and 30 percent by server packet loss." It should be noted, however, that their study focused on a game in the MMORPG genre, which has different player tolerance characteristics than games in other real-time genres, particularly first-person

shooters.

3 Latency Reduction & State Consistency Methods

3.1 Pessimistic Methods: Consistency First

Pessimistic distributed state synchronization methods ensure consistency at the cost of additional delay.

3.1.1 Lockstep Synchronization

Lockstep synchronization requires the game state from all users to be transmitted and confirmed before the simulation clock can be incremented [5]. This ensures complete consistency among the distributed users, at the cost of potentially severe delay. Lock step synchronization is not appropriate for real-time games, because there is no way to guarantee that the simulation will move forward at an interactive speed, especially in a consumer environment. Lockstep simulation is used in certain distributed military simulations.

3.1.2 Bucket Synchronization

In *Bucket Synchronization*, time is divided into fixed length buckets. Update messages are allocated to different time buckets depending on when they were issued by the senders, which requires global synchronization of clocks [6], [18]. To calculate the updated global state, a participant processes all events in the current bucket [6]. By introducing a synchronization delay (inherent in dividing time into buckets), the impacts of network latency and variability is reduced [5], [18].

An optimistic version of bucket synchronization is also available. This algorithm follows the pattern laid out above, but after a set delay, any missing updates are dead reckoned, which creates state inconsistencies [5] (see dead reckoning entry below).

See Local Lag for a similar, but simpler, algorithm.

3.2.6 Artery [4]

Artery was created in response to the high overhead of the IEEE 1278 Distributed Interactive Simulation (DIS) and BBN SIMNET military simulation protocols. According to Chiueh, DIS was designed to ensure full replication and strong consistency, requiring "the entire database of the simulated world to be replicated in each of the participating host[s]". Additionally, "the database states in the hosts are required to be consistent at all times. This comes at a cost to bandwidth: 375 Mbits/sec to support 100,000 simultaneous players (and that was in 1997).

Artery aims to reduce the cost of maintaining distributed state consistency by relaxing the consistency requirements, allowing transient inconsistency. Artery does this by grouping nodes (players & simulation processes) into consistency groups based on spatial characteristics. Nodes can belong to multiple groups. The state among nodes within a group is guaranteed to be consistent, but different groups can be inconsistent. This ensures that a player receives updates that matter to his world state, while reducing the number of messages required to keep the state consistent.

Artery implements *dead reckoning*, described below, and message aggregation to further reduce network bandwidth usage and to compensate for network delay.

3.2 Optimistic Methods: Latency Reduction takes Precedence

Optimistic distributed state synchronization methods execute events before knowing for certain that no earlier events will arrive [5]. This reduces perceived latency, by ensuring that the simulation continues updating, at the cost of a certain amount of inconsistency between different copies of the game environment.

3.2.1 Dead Reckoning

In *dead reckoning*, the next state of each game entity is predicted by all peers by using locally available information, such as position and velocity. A controlling application for each simulated game entity determines if the calculations performed by the distributed peers have deviated too far from the actual state, and if so, the controlling application sends

an update message [4], [12]. Dead reckoning is a “state prediction and state transmission” mechanism [12], [13], which reduces perceived latency and required bandwidth [4], [17], since fewer messages need to be sent.

Pantel and Wolf found that dead reckoning's accuracy can be improved by using input-based prediction methods, where the user's future input is predicted by assuming constant control device position, instead of position-based methods [17].

The drawback of dead reckoning is the loss of some state consistency, which can result in odd situations as the state is corrected. For example, suppose that a car in a racing simulator is traveling in a straight line. A dead reckoning implementation may predict that the car will continue moving forward, and update the local view with that assumption. If the car actually made a turn, the controlling entity will eventually send an update packet, which will cause the car to “warp” to the new, correct position.

Mauve describes a more dramatic situation [13]:

> player A shoots player B. Using dead reckoning the controlling application of player A will create a bullet entity with a certain heading and velocity...Upon receiving the state of the bullet, remote applications will start to check whether any entity that they control is hit... [The] network delay may be much larger (in the order of 100ms or more) than the amount of time that the bullet needs to hit its target. During this time player B might take actions, e.g., shoot at another player C, even though he should not be able to do so (because he's dead).

Despite these drawbacks, dead reckoning has been widely studied [2], [4], [12], [13], [14], [17] and appears to be widely used. It is effective in reducing the perception of latency that comes with playing a game over the Internet, at the cost of reduced state coherency.

3.2.2 Timewarp

Timewarp (also referred to as Time Warp) extends dead reckoning by enabling rollback to a previous, confirmed state when a state inconsistency is detected through the arrival of a new event [13]. Once the rollback has occurred, the event is placed in the queue, and the

events are replayed to generate the current, valid game state.

Briefly, timewarp involves the following [13]:

- Each distributed application periodically saves a snapshot of all simulated entities
- Each application keeps a list of events that is has received
- New events are inserted into the event list as they arrive
- When an inconsistency is detected, a rollback followed by a state re-computation occurs, as described above

Anti-messages are sent as part of the rollback to cancel events which have become invalid. According to Cronin et al., these anti-messages can in turn result in additional anti-messages being generated, from events becoming invalidated, ultimately consuming non-trivial amounts of network bandwidth and overloading the server with the processing load [5].

3.2.3 Local Lag [12]

Local lag works by "deliberately decreasing responsiveness to lower the number of short-term inconsistencies." In other words, local events are delayed to approximately match the latency of the network, up to a certain delay value. If the delay value required to match the network latency would exceed the tolerance of users, local lag can be combined with dead reckoning or time warp.

Local lag is essentially a buffering mechanism, helping to eliminate the effects of high jitter. If a high enough value is selected, most inconsistency can be eliminated (aside from those due to packet loss or extreme delay), but at the cost of causing noticeable lag to the user. At that point, local lag essentially becomes a pessimistic method.

3.2.4 Trailing State Synchronization [5]

Trailing State Synchronization (TSS) was designed to be fast but also ensure strong consistency. TSS runs multiple copies of the game state, each at a slight lag to the next. Only the leading state is shown to the user, while the "trailing states are used to detect and

correct inconsistencies." Each trailing state executes all commands, but on a delay to the next state. If the system determines that its current state is invalid, such as if an event arrives for a previous time that would invalidate the current state, it rolls back to a previous state. The new event is inserted into the event queue, and the newly rolled back state is fast-forwarded to the correct time, resulting in a corrected state. At a high level, this is similar to Timewarp, but at a lower cost in memory (for saved states) and bandwidth (for anti-messages).

This is in contrast to basic dead reckoning methods, which do not attempt to repair intermediate states. However, unlike dead reckoning, TSS requires quite a bit of additional memory to retain the additional states.

3.2.5 Rendezvous [2]

Rendezvous was designed for "high latency collaborative systems, including mobile multiplayer games." It allows short term game state inconsistencies so as to ensure real-time behavior, while ensuring long term consistency by using *Rendezvous arbitrators* and *adaptation rules*.

Rendezvous works as follows: each game node operates asynchronously from all others. Rendezvous arbitrators ensure that each asynchronous game view does not become too disconnected from the others. They do this by generating *target states*, which are points at which the game views can become consistent. It is then up to the adaptation rules to push the local view toward the target state, by controlling simulated entities and subtly bending the rules of the game universe.

Rendezvous assumes that target states will never actually be reached. Instead, they are just a means of ensuring that the distributed views don't become too far out of sync.

According to the Chandler and Finney, one advantage of Rendezvous is that increasing latency has little overall effect on the consistency or playability of the test game (a multiplayer soccer game on a low-bandwidth mobile device) [2].

3.3 A Note on Clock Synchronization

Many of the algorithms listed in this section require the clock for each computer that is sharing distributed state to be kept synchronized within the tens of milliseconds (or less) of each other [6], [13], [16], [20]. The Network Time Protocol (NTP) [23] and Berkeley Algorithm [7] were the two most frequently cited methods for maintaining this property. A full discussion of these methods is beyond the scope of this study.

4 Experiment: Latency Tolerance Testing with *Triangle Wars*

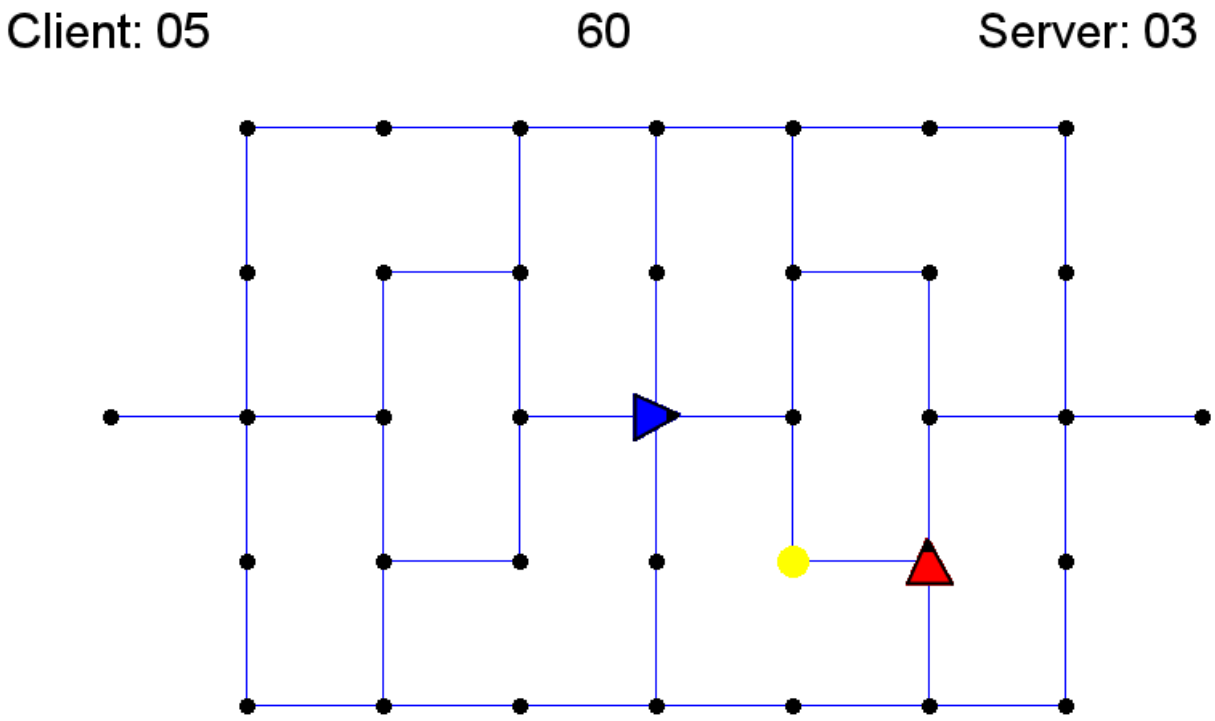


Figure 1: Triangle Wars. The local player is always blue.
The yellow circle is a goal. Once picked up, a new goal will be randomly placed.

Triangle Wars is a networked real-time game that was implemented after reading the literature described in sections 2 and 3. The application can simulate network delay by delaying the send time of packets. The goals of this implementation were:

- Perform preliminary testing on player fairness perceptions at varying network latency levels
- Learn or partially learn how network delay and other impacts can be simulated for

testing purposes

- Create the initial implementation for a system that can be extended in the future to include various algorithms described in section 3
- Implement an application that interacts directly with the Transport Layer, without using an existing intermediary Application Layer protocol, enabling the author to put into practice a small part of what we have learned in class

Triangle Wars was written in approximately 1,700 lines of C++ and uses the cross-platform Simple & Fast Media Library (SFML) [24] to abstract platform-specific graphics, user input and networking system calls behind a consistent object-oriented interface. UDP was used at the Transport Layer, since this was true of all games described in the literature except for MMORPGs, which interestingly frequently use TCP according to Chen, Huang and Lei [3].

The architecture is client-server, with certain peer-to-peer modifications. For example, while the server performs the goal check for all players (i.e., it checks to see if the player has picked up the yellow goal circle, which increments the player's points by one), the client player performs its own movement validity checks. The server never questions the client's position.

Application-level packet headers consist of a one-byte packet type identifier. The data payload ranges from one byte for the smallest packet type to 16 bytes for the largest type (the server-to-client update packet).

Simulated network delay for the client can be introduced at the beginning of each round. This delay is simulated by delaying the send time of the packet to the client by the delay time specified at the start of a match. A fast network link with a round trip time of less than 1 millisecond was used to ensure that any delays encountered were from the simulated delay.

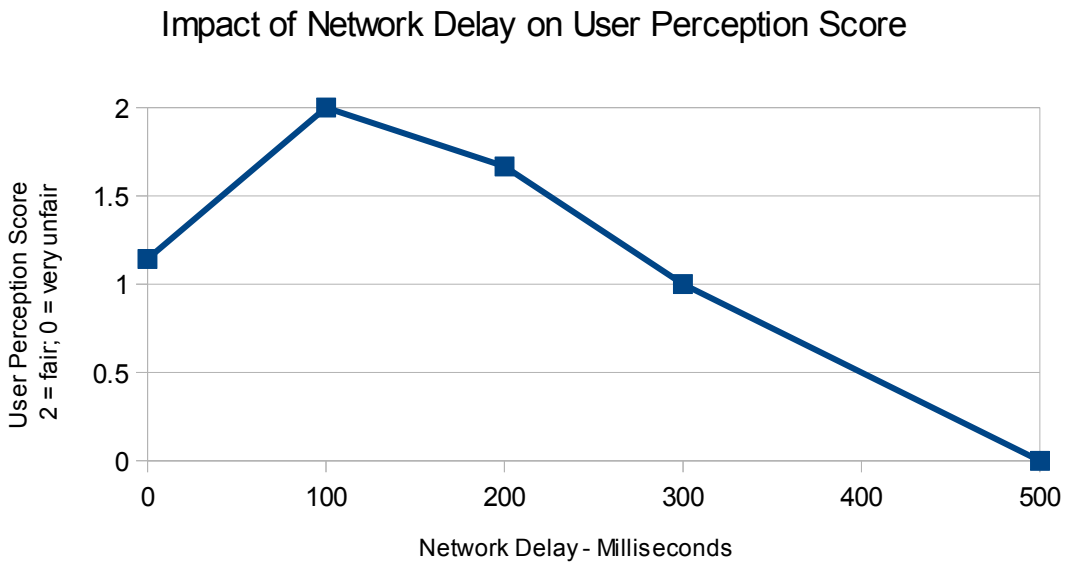
The gameplay was designed to be simple and fast, so impacts of network delay could be readily seen:

- Two opposing triangles start at opposite sides of an arena
- At match start, a yellow goal circle is randomly placed in one of the center waypoints. This is generated by the server player's machine, which gives the server player an advantage if network latency is high, since the client won't know where the goal is. This was by design.
- Players receive one point per goal that they pick up

4.1 Methodology

In order to ensure that the tester was comfortable with the gameplay and control scheme of the game, a series of training rounds was played. The data from the training rounds was not recorded.

Once the training rounds were completed, the user was given a sheet of paper numbered 1 to 20. The user was then instructed to label each game as 2 (fair), 1 (slightly unfair) or 0 (very unfair). Each game used a randomly assigned additional packet delay, which was simulated by the game, of 0, 100, 200, 300, and over 500 milliseconds. The two computers were connected to the same local network using Ethernet cables, and consistently showed a round trip time time of zero milliseconds according to the Microsoft Windows ping utility. For comparison, during preliminary testing, when the machines were connected through wireless connections, average round trip time was typically 60 milliseconds or more.



4.2 Findings

The findings are broadly consistent with the findings described in section 2. Interestingly, a delay of zero milliseconds had a lower User Perception Score than a delay of 100 or 200 milliseconds. This is likely because of the following:

1. The sample size, at 20 games in total, was small, which enabled outlier ratings to have an inordinate impact on the overall rating
2. Although the game is fast paced, it is not as fast paced as a first-person shooter, for example due to the control scheme and the overhead nature. This likely results in a less noticeable delay at the 100 to 200 millisecond range [10], [15], [19]. It may have been difficult or impossible for the user to notice a difference between the values in this range.

As expected, the user noted a large decline in her fairness score starting at a delay of 300 milliseconds. At 500 milliseconds and above, the user always rated the game “Very Unfair”.

It should be noted that these findings are only preliminary, and are not scientifically valid, because of the small sample size. However, the structure used in this experiment could be extended in the future to create more rigorous results.

5 Future Research

Comparison tests of the algorithms listed in section 3 would provide further insight into the benefits and drawbacks of each method described. Now that the initial implementation of Triangle Wars has been completed, one or more of these algorithms should be added to the application, and the perception tests should be re-run. Eventually, it would be interesting to implement a broad cross-section of these algorithms in the application.

Additionally, to increase the scientific validity of any findings, all tests should be re-run with additional users. Once the sample size has been increased, proper statistical analysis can be performed on the results.

Adding the ability to simulate network jitter and packet loss, in addition to the already-implemented constant delay, would enable deeper tests to be performed. This is important since Chen, Huang and Lei found that absolute delay due to the network mattered less to players than delay jitter and packet loss [3].

Finally, performing the perception tests with additional simultaneous players may provide interesting results. For example, it's possible that having more players on the map could create a slight distraction to the user, lessening the perceived impact of network delay. This would require expanding Triangle Wars to include additional simultaneous players.

6 Summary & Conclusions

As described in section 2, network latency can have a big impact on player enjoyment in networked real-time digital games. This is important, since one goal of networked games is to maximize user enjoyment. After all, if a user is having a frustrating experience due to unmasked network latency, why would they continue to spend time (and potentially money) on the game?

Section 3 described a number of algorithms that developers can use to address network latency, at the cost of tolerating a certain amount of distributed state inconsistency. These algorithms ranged from relatively simple and low-cost, such as dead reckoning, to more

complex methods like Timewarp and Trailing State Synchronization, which essentially mimic the behavior of a transactional database.

Section 4 described *Triangle Wars*, which is a networked real-time game that enables simulation of constant network delay at the client. This was used to perform preliminary user perception tests, which were broadly in line with the literature. Now that this application has been implemented, it can be extended to include more complex functionality, such as the latency and state management algorithms described in section 3, as well as the ability to simulate more complex network quality of service impacts, including packet loss and jitter.

7 References

[1] Armitage, G. "An Experimental Estimation of Latency Sensitivity in Multiplayer Quake 3." *The 11th IEEE International Conference on Networks*, p.137-141, September 28 – October 1, 2003.

[2] Chandler, A. and Finney, J. "On the effects of loose causal consistency in mobile multiplayer games." *Proceedings of the 4th ACM SIGCOMM workshop on Network and system support for games*, p.1-11, 2005.

[3] Chen, K., Huang, P., Lei, C. "Effect of Network Quality on Player Departure Behavior in Online Games." *IEEE Transactions on Parallel and Distributed Systems*, vol. 20, no. 5, p.593-606, May 2009.

[4] Chiueh, Tzi-cker. "Distributed Systems Support for Networked Games." *The Sixth Workshop on Hot Topics in Operating Systems*, p.99-104, May 5-6, 1997.

[5] Cronin, E., Filstrup, B., Kurc, A. and Jamin, S. "An efficient synchronization mechanism for mirrored game architectures." *Proceedings of the 1st workshop on Network and system support for games*, p.67-73, 2002.

[6] Gautier, L. and Diot, C. "Design and Evaluation of MiMaze, a Multi-Player Game on the Internet." *Proceedings of the IEEE International Conference on Multimedia Computing and Systems*, p.233, 1998.

[7] Gusella, R., and Zatti, S. "The Accuracy of the Clock Synchronization Achieved by TEMPO in Berkeley UNIX 4.3BSD." *IEEE Transactions on Software Engineering*, vol. 15, no. 7, p.847-852, July, 1989.

[8] Han, Yong-Tae and Park, Hong-Shik. "UDP based P2P Game Traffic Classification with Transport Layer Behaviors." *14th Asia-Pacific Conference on Communications*, p.1-5, October 14-16, 2008.

[9] Hashimoto, Y. and Ishibashi, Y. "Influences of network latency on interactivity in networked rock-paper-scissors." *Proceedings of the 5th ACM SIGCOMM workshop on Network and system support for games*, Article No. 23, 2006.

[10] Kenesi, Z., Kiss, G., Levendovsky, J., Molnar, S. "Optimizing Multiplayer Gaming Protocols for Heterogeneous Network Environment." *IEEE International Conference on Communications*, p.1606-1611, June 24-28, 2007.

[11] Kyung Seob Moon, Muthukkumarasamy, V., Nguyen, A.T.-A., Hyung Soo Kim. "Maintaining Consistency in Distributed Network Games." *13th International Conference on Networks*, vol. 1, November 16-18, 2005.

[12] Mauve, Martin. "Consistency in Replicated Continuous Interactive Media." *Proceedings of the 2000 ACM Conference on Computer Supported Cooperative Work*, p.181-190, December 2000.

[13] Mauve, Martin. "How to Keep a Dead Man from Shooting." *Proceedings of the 7th International Workshop on Interactive Distributed Multimedia Systems and Telecommunications Services*, p.199-204, October 17-20, 2000.

[14] McLoone, S.C., Walsh, P.J., Ward, T.E. "An Enhanced Dead Reckoning Model for Physics-Aware Multiplayer Computer Games." *IEEE/ACM 16th International Symposium on Distributed Simulation and Real Time Applications*, p.111-117, October 25-27, 2012.

[15] Nichols, J. and Claypool, M. "The Effects of Latency on Online Madden NFL Football." *Proceedings of the 14th International Workshop on Network and Operating Systems Support for Digital Audio and Video*, p.146-151, 2004.

[16] Pantel, L., and Wolf, L. "On the impact of delay on real-time multiplayer games." *Proceedings of the 12th international workshop on Network and operating systems support for digital audio and video*, p.23-29, 2002.

[17] Pantel, L. and Wolf, L.C. "On the Suitability of Dead Reckoning Schemes for Games." *Proceedings of the 1st Workshop on Network and System Support for Games*, p.79-84, April 16-17, 2002.

[18] Pellegrino, J. and Dovrolis, C. "Bandwidth requirement and state consistency in three multiplayer game architectures." *Proceedings of the 2nd workshop on Network and system support for games*, p.52-59, May 22-23, 2003.

[19] Ratti, S., Towle, C., Proulx, P., Shirmohammadi, S., "FizzX: Multiplayer Time Manipulation in Networked Games." *Proceedings of the 8th Annual Workshop on Network and Systems Support for Games*, Article No. 11, November 2009.

[20] Savery, C. and Graham, N. "What + when = how: the timelines approach to consistency in networked games." *Proceedings of the 10th Annual Workshop on Network and Systems Support for Games*, October 06-07, 2011.

[21] Webb, S. and Soh, S. "Cheating in networked computer games: a review." *Proceedings of the 2nd international conference on Digital interactive media in entertainment and arts*, p.105-112, 2007.

[22] Yong Woon Ahn, Cheng, A.M.K, Baek, J., Fisher, P.S. "A Multiplayer Real-Time Game Protocol Architecture for Reducing Network Latency." *IEEE Transactions on Consumer Electronics*, vol. 55, no. 4, p.1883-1889, November 2009.

[23] "NTP: The Network Time Protocol." Internet: http://www.ntp.org/ [July 28, 2013].

[24] "SFML: Simple and Fast Multimedia Library." Internet: http://www.sfml-dev.org/ [July 29, 2013].

Appendix A: Notes on Compiling the Code and Running the Game

Triangle Wars was uses C++11 features, and has an external dependency on the SFML libraries and headers. A modern compiler, such as Visual Studio 2012, must be used because of the C++11 features.

I've included the executable for Windows. This should run on any Windows machine, and has no dependencies aside from the font file and image file, which are included in the folder with the executable. All other dependencies (MSVC runtime and SFML libraries) were statically compiled into the executable.

In theory, the code base for Triangle Wars should be fully cross-platform compilable (Windows, Mac OS X, Linux) as long as the correct SFML libraries are downloaded (see link in references). However, I did not have time to test this.

Triangle Wars is secondary to the literature research, though it was fun to implement (if a bit stressful at the end). I would be happy to assist with compiling or running it if you would like, though that is no critical to this project.