5th ISCA/DEGA Workshop on Perceptual Quality of Systems (PQS 2016)
29-31 August 2016, Berlin, Germany

Assessing the Accuracy of Network Estimations in the DOTA 2 Game Client

Matthias Hirth[1], *Fabian Allendorf*[1], *Florian Metzger*[2], *Christian Schwartz*

Chair of Communication Networks, University of Würzburg
Chair of Modeling of Adaptive Systems, University of Duisburg-Essen
matthias.hirth@informatik.uni-wuerzburg.de, fabian.allendorf@stud-mail.uni-wuerzburg.de
florian.metzger@uni-due.de, christian.schwartz@gmail.com

Abstract

Online video games and the subjective quality of player interactions with them rely on good network conditions. Almost equally important is a good and timely knowledge of such conditions in order to take proper countermeasures to worsening conditions. To this end, many online video games include components for estimating the current network status. In this paper, we examine the accuracy of those estimations for the popular competitive multiplayer game DOTA 2. Our results show that the game client is capable of performing a good estimation of the delay and packet loss, but only after a rather large initial delay.
Index Terms: Dota 2, game client, network measurements

1. Introduction

Playing online video games has become a regular recreational activity for many people around the world. Popular game titles are often played by hundreds of thousands of users concurrently, sometimes even millions. This imposes significant challenges on the underlying network and server infrastructure, especially for games with real-time constraints played in a competitive fashion. In these games, lags (i.e., the delay between a player's action and the observable reaction of the game as processed by the server) can cause to disadvantages for the player and subsequently a degraded game experience, ultimately even leading to gamers stop playing that particular game altogether.

While the server infrastructure can usually be directly monitored and controlled by the online game's operator, monitoring the entire transmission path, on which the fidelity of the game strongly depends, may impose some challenges. Additionally, it is generally not feasible to deploy dedicated measurement probes at the players' locations. To overcome such issues many online game clients implement a network monitoring tool in order to estimate players' network Quality of Service (QoS). Such data is then used both to keep the player informed as well as to drive various mitigation techniques modern games are usually capable of. But only if this data is correctly measured and interpreted can the game engine engage appropriate measures.

With that in mind, this study aims to analyze the accuracy of such a network monitoring tool from an exemplary online video game. DOTA 2, a team-based multiplayer game, has garnered a lot of popularity over the recent years, reaching a daily peak of concurrent players of just under $1M$[1]. It is the standalone successor to a custom WARCRAFT III map called DOTA (short for "*Defense of the Ancients*") which started the Multiplayer Online Battle Arena (MOBA) genre. Likewise, it is also one of the most successful competitive games with last year's "The International 5" tournament having a prize pool of over $\$18\,\mathrm{M}$.

This competitive nature makes DOTA 2 an ideal candidate for investigations as undiscovered and untreated QoS variations could have a significant impact on the outcome of a match. In order to examine this situation a dedicated network testbed was set up to test the accuracy of the game's estimation tool under various network QoS conditions, i.e. increased delay and packet loss.

The remainder of this work is structured as follows. In Section 2 we set the background for this work, including a short introduction to the game and its mechanics, and then briefly discuss related work. The methodology used for our analysis is described in Section 3. The results of the conducted studies are shown in Section 4. Section 5 discusses these results and concludes this paper.

2. Background and Related Work

DOTA 2 is a MOBA, where two teams of five players face off against each other. At the beginning of each match, each player chooses a hero as the sole player-controlled unit in the match. During the course of each match, heroes gain access to different combat abilities and skills through a progression system. The system is governed by experience points and currency, both of which are attained through combat with Non-Player Character (NPC) units and heroes from the other team. A typical round of DOTA 2 is about $40\,\mathrm{min}$ to $50\,\mathrm{min}$ long and can be separated into three phases: early, mid and late game. During early game (also called the "laning phase"), the players are slowly ramping up their heroes through static combat with NPC units (or "lane creeps"). While experience points are gained automatically in the vicinity of a dying creep, in order to gain currency one must take the "last hit" on such a creep. This can be a challenging task when affected by lag as timing and precision are required. During mid and late game, the focus shifts to Player-versus-Player (PvP) interactions, with ambushes, skirmishes and larger teamfights. Here, the correct use and timing of one's abilities is crucial to come out on top. Delay and packet loss become even more impactful here when players can miss their correct timings on their abilities.

To keep track of network statistics, the game provides means of testing and observing the current network state. This information can be displayed in the game client as a "*netgraph*", shown in Fig. 1. This DOTA 2 UI widget shows information on frames per second — highlighted as (1) in the figure) — average bandwidth and size of the last incoming (2) and outgoing packet (3), percentage of lost packets (4), amount of delay (5), received updates per second (6), and received packets per sec-

[1] http://store.steampowered.com/stats/ (Jun. 2016)

10.21437/PQS.2016-6

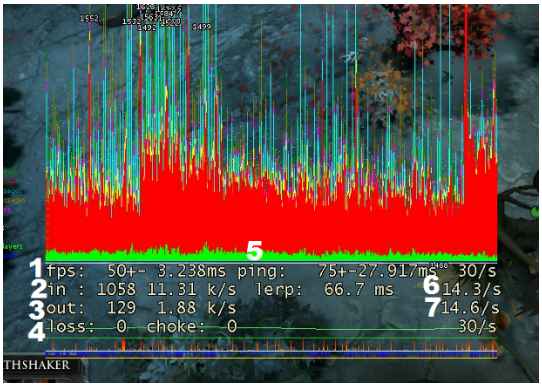


Figure 1: Ingame display of the game client's network measurements, also called the netgraph.

ond (7). Throughout this work, we analyze how accurate the obtained measurement values for delay and packet loss are and how quickly the measurement tool reacts to network changes.

The network's QoS parameters can significantly alter the subjective quality a player experiences when playing online multiplayer games, even if games readily offer a few options to counteract or at least diminish the effects. This includes specific mechanisms like aim assistance [1], but also more general paradigms like interpolation, prediction, as well as lag compensation as outlined, e.g., in [2].

There are numerous studies on the reaction of players to network QoS issues, e.g., [3], [4], [5], [6]. However, all of these studies have been performed in laboratory environments with a limited number of participants. In order to enable large scale user studies, it would be desirable to perform tests with the actual games and use the information from in-game network measurements. To make this possible an assessment of the accuracy of such data is required beforehand.

3. Methodology

To analyze the behavior of the DOTA 2 netgraph, a network emulation testbed was set up, which alters the QoS parameters of the connection between game server and client. In the following, we first take a look at the testbed setup, thereafter we detail the measurement procedure.

3.1. Testbed Setup

Fig. 2 schematically depicts the testbed setup. In order to better control the test conditions we use a local client-sever setup instead of the official online game servers. DOTA 2 version 6.88 is installed both PCs, with the client running Windows 7 64-bit on an Intel Core i7-4790K 4.00 GHz processor with 16 GB RAM. The server is running Windows 7 64-bit on an Intel Core i7-2600 3.40 GHz processor with 8 GB RAM. Client and server are connected through an emulation node equipped with an Intel Pentium 4 3.0 GHz processor, 3 GB RAM and running a 32-bit Ubuntu Server 14.04. The network impairments are generated using NETEM[2]. In addition to the measured link between client and server, each entity is also connected to a separate management network, allowing us to control the experiment without interfering with the actual game traffic. The testbed additionally enables Internet access for the server and client, as this is required to operate DOTA 2.

[2] `http://www.linuxfoundation.org/collaborate/workgroups/networking/netem` (Jun. 2016)

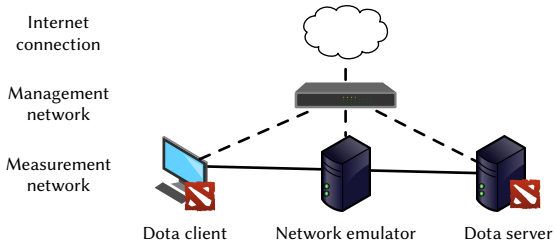


Figure 2: Network testbed entities and interactions.

3.2. Measurement Procedure

The adaptations of the netgraph to the emulated test settings are recorded a during regular game sessions. In order to create game interactions, the players are emulated by bots, computer-controlled heroes, available in DOTA 2. Throughout the game, the network parameters are changed periodically. The emulator holds each parameter setting, e.g., a delay of 100 ms, for two minutes before once again dropping all artificial network impairments for a period of one minute. This allows the game and monitoring tool to reset to the default state before the next QoS parameter is altered.

In order to access the information shown in the netgraph, we used CHEAT ENGINE[3], which allows access to the memory section of other programs, as it is the only means to directly access the raw data the DOTA 2 netgraph is based on. Cheat Engine's memory scanner hooks into other processes and can read and modify their memory area, making it possible to search for specific values. Once the address and offset of a stored value is found, it can be monitored throughout the lifespan of the process. We used this to log the necessary information with a frequency of 21 Hz (i.e., 21 measurements of delay values) given to us by the network monitoring tool.

4. Evaluation

With this methodology at hand, we first have a look at the general behavior of the DOTA 2 netgraph when changing the underlying network parameters. We identify two main metrics to assess the quality of the client's network parameter estimation, namely 1. the *awareness time* and 2. the *adaptation time*, which we will both describe in more detail in the remainder of this section. Finally, we analyze how both metrics behave depending on the emulated delay and packet loss.

4.1. Adaptation Behavior

To gain initial impressions of the accuracy of the game client's network estimation, we use the network emulator to create artificial delays of 10 ms to 2000 ms on the link between client and server.

Fig. 3 shows the delays as measured by the DOTA 2 netgraph and the corresponding network emulator setting. The figure depicts only a subset of a complete match's duration at just one of the considered delay settings, namely 250 ms. The emulator setting, indicated by the continuous line, exhibits the

[3] `http://www.cheatengine.org` (Jun. 2016)

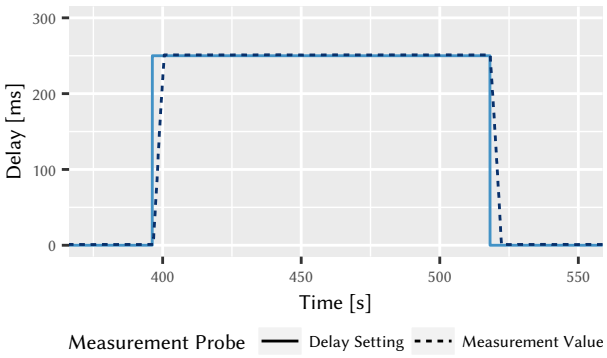


Figure 3: Emulated delay and network condition estimated by DOTA 2 game client.

previously described behavior. Before the delay is increased to 250 ms at the 396 s-mark, the network does not show any impairments. Then the emulator adds an artificial delay of 250 ms for two minutes. Thereafter, the emulator once again removes the additional delay at about 518 s and the link returns to the baseline conditions.

We observe that the game client, shown as the dashed line in the figure, captures the changing network parameters quite well and correctly identifies the additional delay. However, the network state identification does not occur instantaneously. Rather, a transient phase between the update of the emulator setting and the client measurement output can be observed. A closer analysis shows that this transient phase can be further sub-divided into two parts, which we refer to as *awareness time* and *adaptation time*. The *awareness time* describes the time between changing the emulator settings and a first change of the netgraph's measurement, i.e. it measures how long it takes until the client recognizes a change in network state. As soon as a change is recognized, it takes and additional amount of time until the netgraph displays the correct estimation of the current network state. We refer to this phase as the *adaptation time*.

Both *awareness time* and *adaptation time* can have a significant impact on the player's performance and enjoyment, especially if the network conditions change from a good to an unacceptable state. During these two phases the game is not aware of the severity of the network impairments, and therefore cannot take the correct countermeasures, e.g., an automatic pause of the match might happen too late, leading to disadvantages for the affected players. Similarly, if the delay estimation is wrong, lag compensation mechanisms cannot operate properly, as the server adjusts the players' positions to incorrectly guessed lag offsets. In the following we take a closer look on the variations of *awareness time* and *adaptation time* in relation to the emulated delay and packet loss.

4.2. Adaptation to Delay Changes

The assessment of the network delay's impact on the *awareness time* and *adaptation time* is conducted with an added delay of 100 ms to 2000 ms. For each setting we conducted 10 replications and then calculated the client's delay estimation error, i.e. the difference of the emulator setting and the value displayed on the game's netgraph.

An example of the estimation error over time is given in Fig. 4a for a delay of 250 ms. Each measurement run is indicated by a different color, however most of the runs show an identical behavior of the netgraph measurement and therefore overlap. We observe that all runs exhibit an initial estimation error of 250 ms, an expected behavior as the client is initially unaware of the change in network conditions. After the *awareness time* has past, the estimation error starts to decrease linearly for all measurements until it reaches zero after the *adaptation time*. This general behavior is common to all delay settings from 10 ms to 2000 ms. Only the value of the *awareness time* and *adaptation time* changes as shown in the Figures 4b and 4c.

For the evaluation, the *awareness time* and *adaptation time* have been automatically derived using the following approach. We first normalize the estimation error by the delay setting in order to calculate the relative estimation error. Then, we search for the first four consecutive measurements with a relative estimation error smaller than 99 %, i.e. the netgraph starts to adapt its estimation. This timestamp is then used as the *awareness time*. Afterwards, we search for the first four consecutive measurements with a relative estimation error smaller than 1 %, i.e. the netgraph's estimation matches the current parameter setting. The *adaptation time* can then be calculated as the delta between this timestamp and the *adaptation time*.

Fig. 4b shows the mean *awareness time* for each delay setting, including the 95 % confidence intervals, which exhibit a reasonably high accuracy. The figure shows that the *awareness time* increases nearly linearly with the emulated delay. This indicates that the game client only updates the delay estimation after receiving a new packet from the server. Consequently, it takes at least one delay-duration until the clients is aware of the changing network conditions.

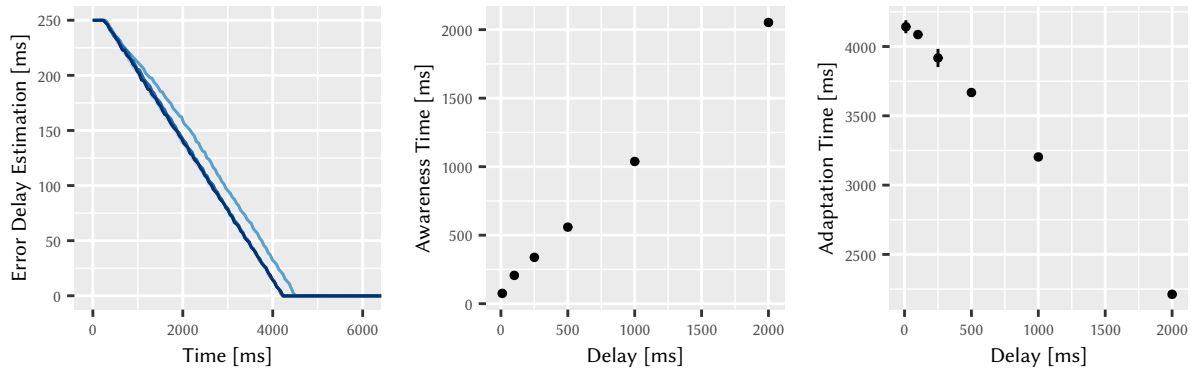
Next, we consider the *adaptation time* as shown in Fig. 4c. Once again, the figure includes the mean values and the 95 % confidence intervals. In contrast to the *awareness time*, the *adaptation time* decreases with the delay. This might indicate that the client uses a simple moving average approach to estimate the network parameters. Thus, a larger change leads also to a faster adaptation.

4.3. Adaptation to Loss Changes

In a second step we analyze the game client's adaptation speed to changing loss between 1 % to 70 %. Similar to the previous evaluation, we performed 10 replications for each setting and calculated the client's loss estimation error as the difference between emulator setting and the value shown on the netgraph.

Figure 5a shows an example of the estimation error over time for 50 % packet loss. We observe a much larger difference between the individual test runs than in the previous delay measurements. This is also the case for all other loss settings and can be explained by the way packet loss is emulated. In contrast to the delay that remains constant for all packets, the packet drop rate follows a random distribution, resulting in larger variations between the different repetitions. These larger variation can also be observed in the values of the calculated *awareness times* and *adaptation times*. The values are automatically determined in a similar fashion to the previous delay measurements.

Fig. 5b shows the mean values and the 95 % confidence intervals of the *awareness time* for the different loss settings. Across all packet loss values, with the exception of 1 %, the *awareness times* seem to be almost constant. This is related to the fact that the server sends packet at a constant rate of about 30 packets per second, allowing the client to detect the changing network conditions for high loss values ($\geq$ 20 %) rather quickly. The large confidence interval and the long *awareness time* for 1 % could be considered as a measurement artifact. Due

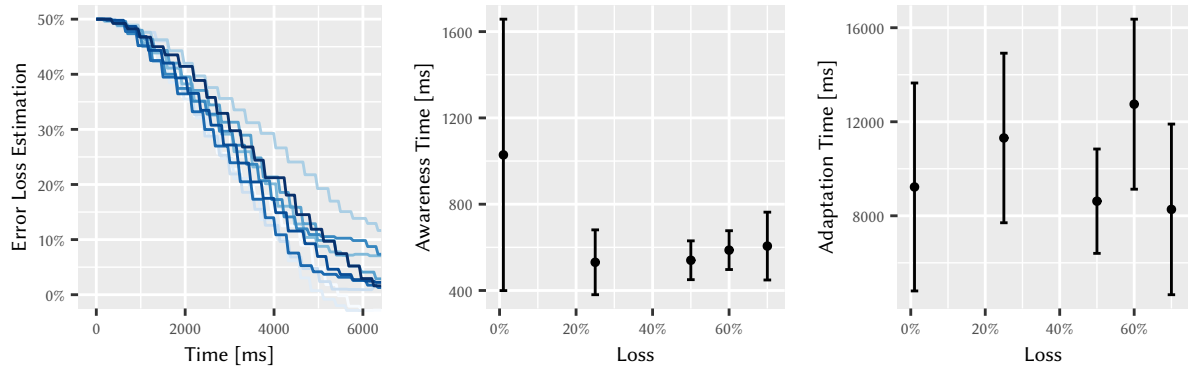


(a) Estimation error for 250 ms delay for multiple measurements.

(b) Awareness time for varying delay settings.

(c) adaptation time for varying delay settings.

Figure 4: DOTA 2 netgraph delay setting times determining the reaction quality.



(a) Estimation error for 50 % loss for multiple measurements.

(b) Awareness time for varying loss settings.

(c) adaptation time for varying loss settings.

Figure 5: DOTA 2 netgraph loss setting times determining the reaction quality.

to the low sending rate and the low loss value, it is quite unlikely that a packet is dropped shortly after the emulator setting is changed. This would make it necessary to calculate the *awareness time* based on the time the first packet was actually dropped rather than based on the changing of the emulator setting. However, this is currently not possible using our testbed.

The *adaptation time* shown in Fig. 5c also appears to be independent of the delay setting, as all 95 % confidence intervals overlap, similarly to the *awareness time*.

5. Conclusions

This paper addressed the accuracy of builtin network measurements performed by online video game clients on the example of the DOTA 2. A proper and timely estimation is necessary for a game to both inform the player about these conditions and perform the correct mitigations. We presented a testbed setup to examine the game client's accuracy of information of varying network conditions and therefore its ability to adapt to them.

The initial evaluation uncovered a transient phase in the game client's estimation accuracy of the current network conditions. This means that both the awareness as well as adaptation to changing network conditions can take considerable time, which can proof to be problematic in certain game scenarios, e.g. a sudden skirmish, where each ability usage has to be correctly timed so that no player can gain an unfair advantage over another. After this transient phase has concluded the client's estimation of the network delay and packet loss are quite accurate, with a relative error of less than 1 % between the emulator setting and the estimation.

These results lay the foundation for future endeavors, as the exact interactions with lag compensation and prediction mechanisms need to be further explored. Additionally, different estimation methods can be evaluated that would enable a faster adaptation to changing network parameters. Likewise, future evaluations need to consider the full end-to-end lag. Besides network QoS disruptions, the full end-to-end lag considers also additional delay factors, e.g., input device and monitor delays but also the game's framerate and tickrate. The latter two can significantly influence the interactions between the player and the game [7].

6. Acknowledgements

This work is supported by the Deutsche Forschungsgemeinschaft (DFG) under Grants HO TR 257/41-1 "Trade-offs between QoE and Energy Efficiency in Data Centers". The authors alone are responsible for the content.

7. References

[1] Z. Ivkovic, I. Stavness, C. Gutwin, and S. Sutcliffe, "Quantifying and Mitigating the Negative Effects of Local Latencies on Aiming in 3D Shooter Games," in *Proceedings of the Conference on Human Factors in Computing Systems*, Seoul, Republic of Korea, 2015. [Online]. Available: `http://doi.acm.org/10.1145/2702123.2702432`.

[2] Y. W. Bernier, "Latency Compensating Methods in Client/Server In-Game Protocol Design and Optimization," in *Proceedings of the Game Developers Conference*, San Francisco, California, USA, Mar. 2001.

[3] T. Beigbeder, R. Coughlan, C. Lusher, J. Plunkett, E. Agu, and M. Claypool, "The Effects of Loss and Latency on User Performance in Unreal Tournament 2003," in *Proceedings of Workshop on Network and System Support for Games*, New York, NY, USA, Sep. 2004.

[4] N. Sheldon, E. Girard, S. Borg, M. Claypool, and E. Agu, "The Effect of Latency on User Performance in Warcraft 3," in *Proceedings of the Workshop on Network and System Support for Games*, Redwood City, California, USA, May 2003.

[5] J. Nichols and M. Claypool, "The Effects of Latency on Online Madden NFL Football," in *Proceedings of the International Workshop on Network and Operating Systems Support for Digital Audio and Video*, Cork, Ireland, Jun. 2004.

[6] M. Jarschel, D. Schlosser, S. Scheuring, and T. Hoßfeld, "Gaming in the clouds: QoE and the users perspective," *Mathematical and computer modelling*, vol. 57, no. 11, Jun. 2013.

[7] F. Metzger, A. Rafetseder, C. Schwartz, and T. Hoßfeld, "Games and Frames: A Strange Tale of QoE Studies," in *Proceedings of the International Conference on Quality of Multimedia Experience*, Lisbon, Portugal, Jun. 2016.