Boom! Headshot!
(Building Neo-Tactics on Network-Level Anomalies in Online Tactical First-Person Shooters)

Mike Bond

Computer Laboratory, University of Cambridge,
JJ Thompson Avenue, CB3 0FD, UK
Mike.Bond@cl.cam.ac.uk

Abstract. This paper tries to de-mystify the urban legends and expose the secrets of success of those who excel at online tactical first-person shooters. It focuses specifically on the evolution of winning tactics, and postulates the existence of a special sort of tactic – the neo-tactic – which exploits the underlying low-level physics properties of the virtual environment. Such tactics may only be employed subconsciously by superior players, or with false justifications for their success. The paper gives examples of individual neo-tactics, such as *first mover advantage* and more complex team neo-tactics; it then suggests experiments that could empirically confirm their significance. The paper concludes that greater understanding and appreciation of neo-tactics could help make online gamers' experiences more harmonious and successful.

1 Introduction

"Boom! Headshot!" is the sound of the Artic Warfare Magnum sniper rifle discharging an 8.6mm round at point blank range, and also the famous maniacal cry of a fictional victorious hardcore FPS gamer playing Counterstrike [2]. Seemingly invincible players are an emotionally evocative segment of the gaming communities surrounding online first-person shooters. They attract persistent attention, are the subjects of both great frustration and admiration, and are frequently accused of cheating. The success of an online tactical FPS is conditional on sustaining interest and support from both the casual community and the competitive hardcore – just like a real-world sport. Thus the "superstars" or heroes of the community have a significant impact in creating the flavour of a game.

For casual players, their play experience when first entering the game will determine whether or not they invest further time and money, and their perceptions of possible unfairness or cheating are crucial.

More serious competitive players will migrate rapidly away from a game if they conclude that fair competition is impossible in light of cheating or game anomalies, and take their funding with them. A hefty proportion of the maintenance and bandwith costs of providing game servers is in the hands of these players, so while they may always have money to spend trialling a game, lack of

support from serious gamers can mean the early demise for the title, and it will appear for sale "on budget" within months.

It is thus interesting and useful to study the secrets of success behind the tiny, most successful portion of players, in the hope that de-mystifying them can promote positive emotional and competitive reactions to their superiority rather than negativity and frustration. De-mystification also rewards game designers for making truly fair games with tactical and strategic depth, as the design of the game environment can more predictably and directly influence the style and social dynamics of play.

This paper first provides a brief overview of the tactical first-person shooter in section 2, setting it in context with other online games, then section 3 describes the internal architecture with particular focus on how network latency is handled, and looks at existing academic research in this area. Section 4 considers a breakdown of the factors affecting success of a player within such a game, and sections 5–7 introduce the theory of neo-tactics, give examples, and discusses impact of such tactics on gaming. Section 8 describes experiments which could be undertaken to explore and confirm the impact of neo-tactics. Section 9 concludes.

2 Online Tactical Shooters

Two characteristics set the online tactical shooter apart from "arcade" first-person shooters. Firstly, weapon physics is more realistic: a single bullet wound to the torso will usually kill, projectiles have realistic arcs and ranges, and rifle accuracy is greatly affected by stance, recoil and motion. Secondly, the theatre of engagement tends to be much larger, sometimes of the order of 25 square km, compared to claustrobic killing fields in arcade games of around .25 square km. Larger battlefields leave more room for tactics and strategy. Empirically, if a competent player kills enemies at a rate below 1 kill per minute, the game is certainly one whose emphasis is tactical.

Modern tactical shooters include Novalogic's Delta Force series, of five games, culminating in the most recent "Joint Operations: Typhoon Rising" edition, and EA Games Battlefield series, with "Battlefield 2" released in mid 2004. The table in figure 1 lists common online FPSes in order of realism.

Individual games typically contain anywhere between 8 and 75 players per team. Games which permit respawning of dead players tend to last between 20 minutes and an hour; those which have permanent death usually have much shorter games lasting only five to ten minutes. Those with large theatres of operations will have vehicle transport – including jeeps, helicopters, boats, tanks and armoured personnel carriers. Some games such as Battlefield 2 [4] attempt to mix infantry-based, armoured and air combat seamlessly on the same battlefield.

Operation:Flashpoint	2002	Closest entertainment product to soldier sim
Joint Operations	1999–2004	Latest in Novalogic's series
Battlefield 2	2005	Latest in Battlefield series, pioneered vehicles
Ghost Recon	2001	Tactical shooters, popular brand on Consoles
Counterstrike	1999–2005	Famous mod to Halflife, most popular online FPS ever
Vietcong	2004–2005	Fairly realistic Vitenam war soldier sim
Farcry	2004	Groundbreaking graphics and vegetation
Unreal Tournament	1999–2004	The pinnacle of the chaotic "deathmatch" play style
Quake IV	2005	Latest iteration of Doom-like arcade shooters

Fig. 1. Popular online FPS games

3 FPS Archictecture & Latency

Nearly all online games currently use a client/server model. The server is likely a rack-mounted PC in a city data centre, and the clientss are home PCs. UDP data packets containing location information for players flow from the server to the client, and movement/firing instructions return in the opposite direction. The server acts on movement information as soon as it receives it, updates its *master copy* of the game state, then relays the new locations of all entities to the clients. However, the presence of latency between client and server complicates matters.

3.1 Client Action Delay

Firstly, the delay between the player causing an action (e.g. firing a weapon, starting to move) and that action taking place creates a vivid and permanent sensation of latency to the user, who expects to begin moving the instant he pushes the forward key. The standard compensation for client action latency is to locally simulate the results of the action. Bernier calls this *client-side prediction* [11]. Given nearly all objects in the virtual environment are immovable, it is very rare for the local prediction to fall from alignment with the server's master copy[1]. On the whole, this compensation works well, but has side-effects we shall see later.

3.2 Target Location Delay

The second effect of client/server latency is that clients receive out-of-date information about the locations of other players, and that clients own interactions with these potential targets (in the form of rifle shots) will be out of date by the time they return to the server. There are three basic approaches to compensating for latency in target information, but unlike client motion prediction

[1] In Joint Operations, the player occasionally becomes stuck when a jump command to navigate a low obstacle such as a sandbag wall gets lost en-route to the server. Server side, the player continues to run fruitlesly at the wall, host side he has completed the jump, moves away from the wall, and then is radically corrected.

(where accurate information about the clients intentions is readily to hand), these compensation effects can often go wrong.

- *Leading aim* – in essence to do nothing. The human attacker himself must predict where his opponent will be by the time his firing instruction arrives; for an enemy running perpendicular to the attackers view, this usually means aiming ahead by an amount proportional to the attacker's latency. Real soldiers must lead aim to compensate for their own physical reaction time, action of the rifle, and bullet flight time. However, the lead aim required to counteract network latency is considerably larger, and the attackers prediction that the oponent will continue his current course may not hold.
- *Extrapolation* – have the client PC perform the prediction as to where the client will be by the time a fire instruction arrives, using a ballistic physics model. The attacker can then aim directly at the enemy, but if the client prediction is wrong, though he may appear to hit, he has actually missed, and the enemy will suffer no damage.

The above two approaches both make aiming a combination of second-guessing and luck, as well as of course the underlying skill required to aim where you wish. They also both share a more substantial disadvantage: *glitching*. In the leading aim case, should any positional update packet be delayed or lost the target will stop moving, and then will blink forward when the new information arrives. In the extrapolation case, if the extrapolation turns out to be wrong the target will blink into a new, corrected position. While this problem is technically no worse than that of accurate firing at the target, the glitching effect is extremely off-putting to the player, and a glitch will occur every time the target changes direction (which could be constantly, should the target be undertaking evasive action). The third approach deals with the above problems, but at a cost.

- *Temporal Buffering* – creates an extra local buffer on the client, and plays the game events to that client after a delay equal to the buffer size. Should packets go missing or be delayed, the client can peek ahead in the buffer and *interpolate* the missing data. The additional delay caused by the buffering in effect increases the client's downstream latency, but of course is only perceived during interaction with the target upon firing a weapon.

These three approaches to dealing with target location latency have their various side-effects as well, especially in combination with other factors.

3.3 Fairness and Latency Compensation

All of the above techniques for dealing with latency have concentrated on creating the illusion of a pleasant and smooth world for the player, and have said nothing about *fairness* in the light of latency. Network latency has been shown to be a clear factor affecting success in arcade online shooters [7, 6]. The basic concept is that if you can perceive a more up-to-date version of the world than your opponents, and your chosen actions will reach the server before the

actions of others, you have both superior information and superior reflexes – a winning combination. An increase in one-way latency of only 50ms creates a measurable reduction in "Quake 3" kills per minute of approximately 30%, according to [9]. Despite the techniques for smoothing the world, focus on making kills keeps competent players acutely aware of latency; Dick et al. reports that some players claim to be able to perceive effects of latency differences as low as 20ms [10][2].

Constructive measures to improve fairness in online FPSes has stayed largely behind the closed doors of industry, though there is some publicly available work, notably from Bernier at Valve [11] and the release of the Quake 3 source code has spurred mod-community efforts such as the "Unlagged" network code stack mod. Aggarwal et al show in [1] that there can be compensation strategies in dead reckoning algorithms to improve fairness even in distributed environments.

Valve's *lag compensation* method [11] (used in both the original Counterstrike and the 2004 Counterstrike:Source games) deserves particular discussion. Lag compensation is basically the temporal inverse of the extrapolation technique described previously in section 3.2. It relies on an architecture where the server can calculate an accurate measure of latency for each client, where all traffic is timestamped, and where the server stores a history of all player locations at times in the past. When a firing instruction is received by the server, the server looks up the state of the game that was used by the *client* at the time the player operating the client fired the shot, and then calculates whether it was a hit. If the shot did hit, the server sends out corrective messages to all affected clients, re-writing the history now that new information has been received.

So while extrapolation looks into the future and guesses where the player will be to assist in aiming, lag compensation *looks into the past*, to see if the shot fired would have hit, if there had been no latency at all. Such a compensating measure clearly creates a new sort of anomaly: because it is no longer the player with the lowest ping that has the definitive say on what happens next in the server, players with good connections will experience world inconsistencies and corrective updates (re-writing of the past) as well as poor players. In an architecture without lag compensation, a player with a perfect connection is guaranteed a consistent and smooth world view.

Note that the fairness methods above are only applicable to balancing high-speed projectile combat[3]. However tactical shooters have extremely brief engagements, where the decisive moment is often who sees the other first, rather than who can aim more effectively. For this reason such fairness algorithms are definitely not a silver bullet. This concludes our discussion of latency effects and

[2] Unfortunately current empirical research on fairness has not properly considered tactical shooters; Counterstrike is the only analysed game which could be classed as tactical, and it possesses few of the more sophisticated elements of tactical shooters like the Joint Operations and Battlefield series.

[3] Again, see Bernier's paper [11] for descriptions of problems for fairness calculations for rockets and missiles

how they are handled; we now focus on the player himself and how his own actions and choices interact with all the other factors to influence success.

4 Skills of a Superhero

A players success in a tactical shooter will be governed by the following factors:

1. quality of play environment, distractions, sound and ambient light
2. specifications of PC, including human interface devices
3. quality of network connection, network hardware, QoS
4. quality of human reflexes and senses
5. experience and conditioned ability in operating FPSes
6. tactical and strategic capability
7. cheating

There is clearly a minimum requirement for a successful player to have an adequate body and mind for the job, to have a adequate PC hardware, and to be familiar with the keyboard/mouse interface at an instinctive level. Whilst ranges of capabilities in these areas may indeed acount for variations in success, the more interesting question is what effect if any the other factors have on success. Section 3.3 describes existing research on linkage of network connection quality to online gaming performance, and cheating clearly has a dramatic effect.

This paper focuses on the remaining area – tactical and strategic capability – as out of all the other categories this is one of the most transferable, and cuts to the heart of what a tactical shooter is suppoed to be about. A spectrum of tactics can be envisaged, ranging from dependence on the real world to dependence on the virtual world.

- Military tactics
- Neo-tactics
- Game world tactics
- Exploits

Military tactics are those from the real world which can be applied to the virtual world[4]. At the other end of the scale, *exploits* are tactics based upon a fault or unintended feature of the game environment, for instance being able to shoot through a brick wall whilst remaining invulnerable, due to over-simplified collision detection algorithms. *Game world tactics* are those which are peculiar to the design of a particular game and divergent from real life, but which are considered to be within the spirit of the design. For example, in Novalogic's Joint Operations game, a fall from any height into deep water will never be fatal, so a common game world tactic is to leap from helicopters into water at great height; this a substitute for the lack of parachutes in the game implementation. The final

[4] Indeed an element of the appeal of online tactical shooters is that they are worlds in which military tactics and strategy are prime influences on success

category is the *neo-tactic*: it is in essence a tactic whose effect is founded the low-level physics properties of the game world, rather than the high-level rules (which are either explicitly written or easily testable). We will now consider neo-tactics.

5 Basic Neo-Tactics

Neo-tactics[5] are not the obvious exploitations of the gap between a simulation and the real world. For example, if one can survive a ten metre fall with impunity, a good tactic may well be to attack by leaping from buildings onto the enemy – such a method is a game tactic, because it is founded on an easily tested, or in fact clearly documented feature of the game world. Neo-tactics are based on the more subtle characteristics of the game physics, to do with network latency, dead-reckoning, or the way that certain game events are processed differently from others.

Moderately experienced players will have some appreciation of the network effects themselves, and through a combination of teaching, logic and direct experience of failure modes, they will have some rules of thumb. For instance, any online FPS gamer worth his salt will surely tell you that *"low ping is better"*. However such a gamer is much less likely to be able to describe how he adapted his gameplay style to make use of his advantage – the realm of the neo-tactic. Furthermore, such acknowledged wisdom may not actually be true: Dick remarks in [10] that while some games behaved as expected and reported higher kill per minute rates with reducing latency, Counterstrike (the only tactical shooter analysed) reported the opposite effect. If sub-optimal network conditions can be to a players advantage, what is the mechanism? This lends some broad credence to the idea that tactical exploitation of network effects (or of in-game fairness algorithms) is at least possible[6].

We now analyse some simple network effects in turn, and look at possible neo-tactics which could be developed to exploit them. When we need to use a specific example (for instance to discuss transaction stream mixing), we refer primarily to Novalogic's "Joint Operations" [3] tactical shooter. Aside from the author's familiarity with the game series, Novalogic games are a relevant and worthwhile choice as they have the longest running series of tactical shooters, and after their early success in the entertainment market, the same technology was used in the LandWarrior soldier simulation for the US military [5].

5.1 First Shooter Advantage

Consider an online tactical shooter game running on a client-server architecture as described in section 3. Two equally capable opponents face off from opposite

[5] Neo-tactics are so named not due to their novelty, but after the character from the Wachowski brothers' film "The Matrix". In this film the lead character, Neo, discovers that he lives in a virtual world and gradually learns to manipulate this world at a fundamental level.

[6] See section 5.2 for the full discussion of this issue

sides of an open field, separated only by a screen of smoke which prevents each from locating the other, as shown in figure 2. When the smoke clears (a game-event we assume to happen simultaneously for both parties), they must locate one another, take aim, and fire.

Fig. 2. Two players separated by a dispersing cloud of smoke

First shooter advantage is the advantage conferred on the player with the lowest upstream latency, as his shoot instruction will reach the server first, thus stasticially it will be him that makes the kill. Should the global event be unpredictable (and thus cannot be relayed to the clients in advance of its occurence), then downstream latency will be an additional factor, and it will be the player with the lowest total latency who has first shooter advantage. We call this scenario *reactive* first shooter advantage.

The architecture might attempt to ameliorate first shooter advantage, for instance by employing lag compensation (see section 3.3). Lag compensation assumes that players fire as a result of a server-sourced stimulus, so compensates for both the latency in stimulus arrival and in transmission of the response. In the case of reactive first shooter advantage, both players are reacting to the same event so the result will be fair – that is, dependent on which human has the better reactions and aim.

In the presence of lag compensation, raw first shooter advantage actually falls to the player with the higher ping, as they are assumed to be reacting to an in-game stimulus and their firing instruction is falsely backdated by both their downstream latency and upstream latency.

Assuming the game does not employ lag compensation, how can a player adjust his gamplay to make use of first shooter advantage? Ironically, first shooter advantage – while it is the most widely appreciated online gaming network effect – is a pretty rare event. One could imagine neo-tactics that exploit global synchronised events, for instance deliberate creation of a smoke screen, or maybe restoration of power to a building which puts the lights on. However, given one player tends to be the actor and the other the responder, that is, events in the game *are caused by external actors*, it is rare to encounter a truly synchronised

trigger event which both players must react to on an equal footing. This brings us to a second network effect.

5.2 First Mover Advantage

Now consider a different scenario: two equal opponents are facing up in an urban environment, hiding from each other around the corner of a building, as shown in figure 3. Both are initially stationary. Player A chooses of his own accord to burst out, aims and shoots at the other; meanwhile B defends himself.

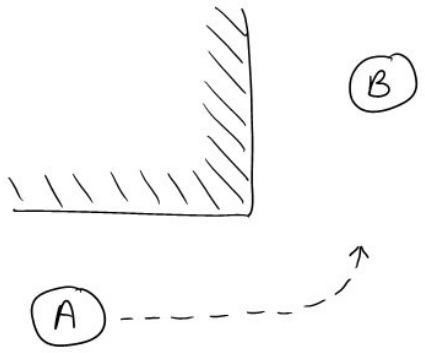

Fig. 3. Two players face off around a corner

Here the attacker, player A, gains *first mover advantage* and will statistically make the kill. This is because B is stationary, so accurate information about his location is already available to A's client, and client motion prediction means this can be displayed immediately to A as soon as he rounds the corner. Meanwhile player B has to wait until this scenario is played out from his perspective. The delay before B reacts will be a sum of the following factors:

- Upstream latency of A to Server (∼100ms)
- Server processing delay and frame boundary rounding (∼25ms)
- Downstream latency of Server to B (∼100ms)
- Temporal Buffering Delay (∼200ms)
- Upstream latency of B to Server (∼100ms)
- Server processing delay (∼25ms)

The example is annotated with some worst case figures, describing two players with relatively slow (200ms ping) broadband connections, yet where player A get a 0.55 second advantage over B. This is a long time in a twitched based shooter: typical human reaction time to visual stimulus is 200–250ms, and a

typical reaction and aiming task (from Fitt's law experiments) takes around 500–1000ms.

It is impossible to totally compensate for first mover advantage, as it fundamentally damages interactivity between the players. Lag compensation can in theory ameliorate all the delay effects by backdating the firing instructions – turning it into a contest of reaction times. A thus initiates a challenge, bursting out and shooting, and sets a target time within which B must react. Later on, B receives this "recorded footage" and tries to beat A's time in the same game. This should in theory be fair, but note two things: A sets the nature of the challenge, and B can only react by shooting, not by evading – for evasive manoeuvres cannot be backdated. Several neo-tactics follow as a consequence:

- Always try to be the first mover
- First mover advantage works best against high-latency players
- If you are caught by a first mover, do not try to evade, immediately go for a kill with the rifle. Setting into motion immediately worsens your rifle aim, yet the evasion instructions are unlikely to arrive before the engagement is over.

Note again that we are leaving aside military tactics, such as the element of suprise that A may possess, and the advantage of concealment that B could have in this engagement.

The 0.55 second advantage can be increased even further due to lag compensation. Lag compensation assumes that player A is acting on an external stimulus when shooting B, so the shot will be backdated by the round trip latency of player A. This could add a further 200ms to the effective advantage, and make evasive manoeuvres for B impossible.

First mover advantage can possibly explain a tactic some FPS players have developed a to navigate an un-secured corner. They do this by approaching, then turning sideways to face the wall, and strafing sideways into view around the threshold. Conventional justification could put this tactic down to bring the rifle's sight to bear as close as possible to the enemy's predicted location. This justification is potentially contradictory as it requires a whole mouse stroke (a start of motion and an end) to bring the rifle to bear, rather than simply ending the stroke that was used to swing round the corner (NB: CHECK THIS). Maybe it is first mover advantage that players the greater role?

5.3 Semi-Auto Advantage

Most rifles in tactical shooters support semi and full-auto firing modes. Players naturally work under assumptions about the appropriateness of these modes drawn from real life considerations, such as accuracy and recoil. However, there are neo-tactical influences too. During automatic fire, the client does not send fire instructions for the full number of bullets expended. Typically for every 5 bullets fired in automatic mode, 3 will be perceived by the server, though the first bullet always counts. Neo-tactically, it is more ammunition-efficient to

use semi-automatic fire. However this is a minor consideration: the much more sinister effect is how semi and full auto fire instructions may be affected by the underlying network.

Considering that there may be substnatial delay between one player setting out on a course of action, and his opponent responding, it can be helpful to think of a tactical shooter contest to consist of a series of brief engagements, each initiated by one or other party, recorded as a video clip challenge, and mailed to the other to react to. Clearly interactivity here is more turn-based than real time – the reaction and aim time of one player set as "the standard" in the video clip must be beaten by the other.

Now consider the effect of latency jitter, where although the average latency may be steady, packets are repeatedly bunched up together – clustered – such as would be done by a cheap ADSL modem. This effectively compresses the video clip challenge so that all the actions within it are played out much more rapidly, setting a much harder time to match. Proper timestamping of packets is required to re-play them at the appropriate rate; if the packets are delayed so that they can be played out at original speed, there is a danger that a particular client will build up a large buffer of pending actions, so will perceive bad lag effects. One effect of packet clustering is to make first mover advantage more pronounced.

The more serious effect may arise due to a fundamental difference between automatic and semi-automatic fire packets. An automatic fire packet in Joint Operations dispenses three bullets with a pre-defined time interval between bullets. Such a packet thus takes time to enact and dispense before the next can be processed, this is the nature of automatic fire implementation. However semi-auto fire results in individual bullet packets, which can be acted on an disposed of next to instantly. Semi-auto fire instructions are thus much more amenable to being time-compressed than auto fire; this architectural quirk amplifies the unfairness due to packet clustering, should the attacker learn the neo-tactic necessary to exploit it.

6 Advanced Neo-Tactics

Complex neo-tactics can arise from a combination of low-level network effects and pathological cases of the algorithms used by game developers to try to ameliorate the problem. In the following examples we consider neo-tactics built around network effects arising from bandwidth limitation.

6.1 Dispersed Defence Advantage

Suppose each player in a tactical shooter has sufficient bandwidth to receive updates on the locations of ten players at the maximum possible rate of 20 updates per second. It clearly makes sense to prioritise which subset of the location data of all players is sent to each individual. A sensible idea could be to assign priority to the closest N players, and the rest with some sort of best effort strategy, which updates at reduced rate, and uses dead-reckoning to fill

the gaps. This particular algorithm is hypothetical, but is founded in real design: in Novalogic's Joint Operations that players at ranges of over 1km have their location updates drastically dropped in frequency.

A whole range of team neo-tactics result from this, running under the assumption that a less frequently updated player will be harder to detect and hit than a rapidly updated one. Suppose one team of players is clustered round a particular objective which they must defend in order to win the game. Seeing as they are all in close proximity, much of their bandwidth is already being used sending rapid updates of each other's own location. Thus the flightpath of an incoming helicopter will be largely dead-reckoned, so harder to hit with anti-aircraft defenses. The neo-tactical avantage here is to attack a clustered defence boldly and at high speed. Conversely, if the opposition form a wide perimeter, each is more likely to get accurate positional data on the incoming threat, and be able to take it down. In this case, the defenders gain a *dispersed defence advantage* (which is completely separate from the military issues as to whether troops are better grouped or dispersed).

6.2 Covering Fire Advantage

The bandwidth limitations exploited by dispersed defence advantage could equally be applied to fire instructions. It may well be that heavy mounted weapon fire into the approximate area of the enemy also wastes their available bandwidth, degrading the remaining data which they receive. This might, for instance, permit an approaching helicopter to make a safe landing. Alternatively, as incoming fire that does not create a hit is considered an "environment update message" and is not re-sent, if someone fires near you (but does not hit you), the arrival of this very threatening shot could be dropped. The crucial fire is thus effectively masked by the covering fire, even though the sound of the covering fire is not necessarily audible.

6.3 Quantised Approach Advantage

Typically the area of specific coverage afforded to a player group will not be calculated via a true radius of interest, but via an approximation to a circle, such as a bounding box, or set of marked cells [8]. It is thus to the attacker's benefit to approach parallel to the axes of calculation – that is, from the compass points – to minimise his time of exposure to detailed enemy observation. Such effects may only be relevant for really high speed vehicles such as jet fighters in Battlefield 2. Interestingly quantised approach advantage occurs in real life military tactics for different reasons – for instance, in the form of attacking from out of the sun, or from above which often takes humans by suprise.

6.4 Ballerina's Advantage

The Ballerina's advantage is to jump; in many arcade shooters this confers an evasive advantage, or even a speed advantage, but the slightly improved realism

of tactical shooters discount the above advantages. However, in Joint Operations, jumping instructions are regularly discarded by the client, rather than being reliably sent to the server. Empirically, for every five or so jumps, only three will be sent: this means the enemy does not always see you jump. However you still gain the visibility and aiming benefit of jumping.

A bullet is essentially modelled as a beam: when you shoot, you transmit a start point for the bullet, and a direction to fire in. This start point is calculated based on where you actually are according to the client. Hence the bullets always come directly from your perception of where your gun is. However the hit is calculated on where the target is according to the server, not based on your perception of where the target is.

A possible neo-tactic, bordering maybe on an exploit, is for an attacker to jump out of cover and fire his rifle, or launch a ballistic weapon. The projectile will never collide with the object used for cover, and there is a 2/5ths chance it can be launced without ever exposing oneself to enemy fire. The impairment of aim resulting from this tactic makes it impractical for use on all but point blank targets, but it is clearly a viable approach for engaging, say, heavy armour which has totally infiltrated a base camp. Alternative engagement approaches of popping up (see section 7.2) or inching around a corner (section 7.3) both have their drawbacks.

7 Countering Disadvantages

Sections 5 and 6 have considered *advantages*, yet there are some activities which are neo-tactically a bad idea. We now consider these *risks*, and show that neo-tactics can provide a convincing argument to avoid using these tactics. To see the examples, we need to learn a little more about the classification of in-game actions, however.

Within the architecture of Joint Operations, the communication of different game-environment actions and events are handled in different ways. We define *events* as things that happen within the game, and *actions* as events which occur in response to human input. Actions can be carried out *optimistically*, *reliably*, *transactionally* or in *hybrid form*.

Optimistic and *reliable* actions are those where the client performs the action as soon as it is instructed to, and thus it will only be later that the central server is told it has been done. Such actions include firing rifles and special weapons[7], certain changes of posture such as leaning and turning (turning is defined here as controlling where you look), bringing weapons to bear, changing weapons, and jumping into the air.

Meanwhile, *transactional* actions are those that require a server round-trip before their effect is visible to the client. These include change of stance between standing, kneeling and prone, claiming control of tactical objectives, entering and exiting vehicles, triggering explosions, taking damage and dying.

[7] Anti-tank and anti-aircraft missiles, claymore anti-personnel mines, grenades and explosive charges

Optimistic actions are those performed locally whose loss or supression to the global view is not considered crucial to the game experience; reliable actions are those that are performed immediately locally, but will be re-sent to the server repeatedly until they are acknowledged; transactional actions will only be performed once a communication round trip has occurred.

Hybrid actions are those whose physical representation to the user seems to be a single action, but that are composed separately of transactional and non-transactional elements.

7.1 Navigation Risk

The crucial hybrid action is *moving*. When you move you send relative movement instructions to the server e.g. "move forward 10 metres". Your client then sets you moving immediately and predicts where you will be. Once the server receives your move instruction it then tells you where you have reached by your move instruction. We call this *arriving*. If your latency is too high, you will arrive somewhere different to where your client predicted you had moved, and this results in visible jitter on your own position. In heat of battle, inability to rapidly and effectively negotiate around obstacles, doorways and vehicles on the battlefield is a crucial shortcoming – a *navigation risk*.

With some thought and understanding, a solution becomes apparent. Strafing – the act of sidestepping – is processed immediately by the server (i.e. optimistically), whereas turning is processed transactionally. If your latency is high you will get the experience of facing in one direction but moving (seemingly side-stepping) in the other. This effect is greatly magnified in high speed vehicles (helicopters and jeeps), and momentary jitter can result in wrong turns or crashes. Therefore in conditions of lag, using a combination of strafing and moving rather than turning and moving allows you to navigate obstacles with less risk of getting stuck.

7.2 Pop-up Risk

As previously discussed, changes of posture are implemented as transactions in Joint Operations. Consider an attacker lying prone behind a sandbag wall. He hits a key to stand up, rapidly brings an RPG to bear on some enemy armour and fires, hoping to catch it by suprise. Unfortunately, the server processes the change of posture as soon as it is received, whereas the effects of the posture change – the ability to aim and fire – are only conferred after the round-trip. Thus any enemy with a lower downstream latency than the attacker will gain an advantage, and extra time to eliminate the threat. The attacker would do much better to crawl away, then run out standing from behind a corner, than risk a posture change; we call this *pop-up risk*.

Attackers lying prone on distant hilltop are usually well camoflagued and difficult to spot. Yet should they come to their feet before advancing forwards they leave a vital period of exposure where they are standing and stationary.

Thus if time permits, a better neo-tactic to negotiate the crest of a hill is to crawl back into cover, stand, then run in over the crest at speed.

7.3 Concealment Risk

Joint Operations also quantises the player's position to the nearest 0.5m or so before an update is sent, even though the resolution of location displayed by the client is much higher. Thus if an attacker inches out from behind a tree in order to get a shot off at someone, he may not move at all on the enemy's screen, then suddenly you will move all in one go and be totally exposed. The attacker has a *concealment risk*, that could conceivably be to his advantage, if he can fire from relative impunity, but given that he cannot measure his server-side exposure, is generally a liability.

8 Experimental Confirmation

It is not easy to test if neo-tactics have significant impact on success rates in online tactical shooters, though it's likewise hard to trivially discount them: the level of experience and teamplay at which they come into effect simply isn't attainable during playtesting of a game. Thus neo-tactics cannot be iteratively debugged out of a game. Furthermore, some neo-tactics are only manifested in situations with co-ordinated teamplay, and existing academic studies have only covered large public access servers.

Online first person shooters are in fact very difficult environments to perform *any* experiments in, because the very nature of the design is to create an illusion – to maintain the appearance of a harmonious and consistent world shared by everyone. The first fundamental assumption of science is that our senses present an accurate and consistent picture of the world around us, and this assumption is clearly and maliciously undermined. Such games hope that side-effects will not cause manifest problems, and the focal gameplay can continue around proper tactics and skills. This grand deception may in part explain the enduring confusion and frustration that many players feel when trying to get to grips or improve their performance in these games. The conscious mind of a player comes up with all sorts of plausible (but unfounded) explanations to explain the strange anomalies that are witnessed regularly subconsciously, but only rarely are substantial enough to break through to conscious level. Thus the player whose subconcious has been saying *"I swear I shot first"* for 20 minutes will have an exaggerated negative response in the rarer cases where a substantial latency or jitter event occurs.

8.1 Confirming Existing Neo-Tactics

Two possible initial exploratory approaches are to use traffic shaping network bridges to create exaggerated network effects, and to examine primarily high-speed vehicle interactions, where the fallout from network effects is visually

clearer. Traffic shaping bridges have their limitations as the real distributed network needs to be modelled very accurately to be sure no peculiar effect is left out. Despite the difficulties with the environment, with the right equipment to synchronise recordings, and a correctly designed experiments to hold other factors constant, it should be possible to collect experimental evidence of the neo-tatics described in sections 5–7. **Todo:** Mention first mover advantage experiments. We now consider specific network-level effects, and propose experiments to explore each. **TODO:** It is possible that the radius of threat for claymores changes depending on how high your ping is. This might just be my imagination, need to do more experiments.

8.2 Discovering New Neo-Tactics

The ten expample neo-tactics discussed are clearly not exhaustive, and considering that they exploit only several of the known types of network anomaly that may occurr, it seems reasonably to assume that there are more. It is conceivable that neo-tactics exist for even very peculiar network effects, and that some come into effect only in multi-party scenarios, rather than the two player examples primarily discussed here. Table 4 summarises broad types of network effects and section numbers for corresponding neo-tactics.

Situation	**Neo-Tactics**
Low latency	Section 5.1
High latency	Sections 5.2, 7.1
Latency jitter/packet clustering	Section 5.3
Asymmetric latency	Section 5.2
Packet re-ordering	None known
Limited bandwidth	Sections 6.1, 6.2
Algorithm approximations	Sections 6.3, 6.4, 7.3
Transactional actions	Sections 7.2
Optim/Tran remixing	None known

Fig. 4. Network effects and corresponding neo-tactics

9 Conclusions

The crucial observation of this paper is that evolution of tactics on top of low-level physics anomalies serves to magnify them. Whilst the game developers may be aware of the simpler problems, and come up with countermeasures to deal with them, the countermeasures themselves can cause further imbalances. Playtesting to deal with neo-tactical imbalance is very difficult, even if issues can hypothetically be detected, the time required for them to manifest themselves

naturally and to identify them is prohibitive. We thus need to learn about them and find a way to *"reason them out of existence"*.

From the point of view of game designers, neo-tactics remain undesirable at a much simpler level: they create the impression of cheating, and no-one enjoys an unfair fight. In the long-term, more advanced protcocols can hopefully fix a lot of the issues without damaging fluidity of the game, in the short time, exposure of the techniques levels the playing field, and reduces accusations of cheating, which damage the reputation and longevity of a game.

From the point of view of players[8] developing understanding of low-level physics tactics is a substantial hurdle in improving an indvidual's ability, but the very poor *transferability* of such tactics is an even greater barrier to improving the ability of a team.

Online tactical shooters are now maturing as a genre, and it's time the games industry and the academic researchers looked beyond the simple network protocol cases toward understanding the more complex effects. There is little point continually developing ad-hoc compensating algorithms for low-level effects if even the tiniest imbalance will be inevitably magnified until it becomes a problem.

References

1. S.Aggarwal, H.Banavar, S.Mukherjee, S. Rangarajan, "Fairness in Dead-Reckoning based Distributed Multi-Player Games", NETGAMES 2005, `http://www.research.ibm.com/netgames2005/papers/aggarwal.pdf`
2. "Boom! Headshot!", Interview with an FPS Gamer by Doug, `http://media.putfile.com/FPS-Doug`
3. "Joint Operations: Typhoon Rising", Novalogic Inc., `http://www.novalogic.com/games.asp?GameKey=JOTR`
4. "Battlefield 2", EA Games, DICE, `http://www.battlefield2.com`
5. Novalogic Corporate History, `http://www.novalogic.com/corp_history.asp`
6. G.Armitage, "Sensitivity of Quake 3 Players to Network Latency", Poster session, SIGCOMM Internet Measurement Workshop, San Francisco, Nov 2001
7. S.Zander, G.Armitage, "Empirically Measuring the QoS Sensitivity of Interactive Online Game Players", Proc Australian Telecommunications Networks and Applications Conference (ATNAC 2004), Sydney, December 2004
8. J. Smed, T. Kaukoranta, H. Hakonen, "A Review on Networking and Multiplayer Computer Games", Turku Centre for Computer Science (TUCS) Technical Report No 454, April 2002.
9. Ubicom Inc, "OPScore: A Metric for Playability of Online Games with Network Impairments", `http://gamer.ubicom.com/pdfs/whitepapers/IP3K-DWP-OPSCORE-10.pdf`
10. M.Dick, O.Wellnitz, L.Wolf "Analysis of Factors Affecting Players. Performance and Perception in Multiplayer Games", `http://www.research.ibm.com/netgames2005/papers/dick.pdf`, NETGAMES 2005
11. Y.W. Bernier, "Latency Compensating Methods in Client/Server In-game Protocol Design and Optimization", Valve Inc.

[8] I won't deny I'm interested in improving my own gameplay!