2009 IEEE 34th Conference on Local Computer Networks (LCN 2009)
Zürich, Switzerland; 20-23 October 2009

Reliable Transmission Over Covert Channels in First Person Shooter Multiplayer Games

Sebastian Zander, Grenville Armitage, Philip Branch
Centre for Advanced Internet Architectures (CAIA)
Swinburne University of Technology
Melbourne Australia
{szander, garmitage, pbranch}@swin.edu.au

***Abstract*—We propose and evaluate a novel improvement to a previously published, unreliable covert channel based on the network traffic of multiplayer, first person shooter online games (FPSCC). Covert channels typically embed themselves within pre-existing (overt) data transmissions in order to carry hidden messages. FPSCC encodes covert bits as slight, yet continuous, variations of a player's character's movements. These variations are visually imperceptible to human players, yet occur frequently enough to create a low bit-rate covert channel. The nature of first person shooter network protocols means the original FPSCC channel is noisy (not reliable), experiencing a significant number of bit errors (including synchronisation errors). We have now augmented FPSCC to ensure bits are transmitted reliably. Evaluation of our technique with a prototype demonstrates throughput of up to 13 bits/second without any bit errors.**

***Index Terms*—Security, Covert Channels, Online Games**

I. Introduction

Unlike encryption (which makes communication between two parties hard to decipher by a third party), covert channels aim to hide the very existence of communication between two parties. Adversarial relationships between individuals and groups can provide various motivations for the use of covert channels. Examples include dissenting citizens versus their governments, government agencies versus criminal or terrorist organisations, or 'black-hat' hackers versus company IT departments. Many network protocol-based covert channels have been proposed [1], with limited work on covert channels in networked games primarily focused on board games [2], [3].

In [4] we proposed a novel covert channel between players in multiplayer first person shooter (FPS) online games (known as FPSCC). FPSCC hides covert information in minute, additional movements of player characters in the virtual world. Character movements dictated by a human player are subject to slight variations to covertly encode additional information. The variations are chosen such that other human players perceive no visible effect.

FPS games are common and their traffic is generally not suspicious. They are typically based on the client-server architecture, and FPSCC channels operate through regular, unmodified game servers. As thousands of public, independently operated game servers may be active on the Internet at any one time [5], FPSCC's users have many servers through which to establish legitimate-looking overt traffic flows.

FPSCC has a number of desirable properties. FPSCC is an indirect channel – covert sender and covert receiver use a game server as an intermediary, rather than directly exchanging IP packets. Detection of the covert sender does not directly expose the identities of the covert receiver(s) (who could be any of the players online at the same time). Some network-based covert channels can be eliminated by, for example, protocol normalisation [1]. FPSCC cannot be eliminated without eliminating player movement (thus destroying the game).

FPSCC could be used for collusion as well as exchanging game-unrelated information unbeknownst to adversaries. Capable adversaries could easily detect the use of overt communication (e.g. instant messaging or voice over IP) as well as the game-internal chat. Beyond games, many companies are exploring the use of immersive virtual worlds (such as Second Life [6]) for distributed training, collaboration and general business – this opens up the potential for covert ex-filtration of commercially sensitive information via FPSCC-like channels.

However, FPSCC only provides unreliable modulation and demodulation of covert bits – the covert channel experiences both bit substitution and bit synchronisation errors (bit deletions and insertions). We now propose Reliable FPSCC (RFPSCC), an enhanced version of FPSCC tailored to the characteristics of FPSCC's overt channel to provide reliable data transport. Furthermore, we propose an information-theoretic model to estimate the capacity of an FPSCC channel

We use the open-sourced Quake III Arena (Q3) game [7] for our proof-of-concept RFPSCC implementation (Q3 runs under Windows and Linux and has a client-server communication architecture similar to many other FPS games). Empirical tests involved a large number of games and various degrees of latency and packet loss impacting on the overt traffic between game clients and server. RFPSCC is shown to be reliable (no bit errors) and exhibits throughput ranging from 2 bits/s to over 13 bits/s. Like many covert channels RFPSCC has a low bit rate, but it is sufficient for SMS-style text messages.

Section II reviews covert channel concepts and first person shooter game protocols. Section III reviews the original FPSCC encoding scheme and its limitations. Our novel Reliable FPSCC is then presented in Section IV. Section V reports on the empirically measured throughput of RFPSCC, develops an information-theoretic channel model and compares the

978-1-4244-4487-8/09/$25.00 ©2009 IEEE

measured throughput with the channel capacity. Section VI concludes the paper and outlines future work.

II. Background

A. Covert Channels

To summarise the discussion found in [1], covert channels are often illustrated by Simmons' *prisoner problem* [8] (Figure 1). Prisoners Alice and Bob must communicate to establish an escape plan, but Wendy (the warden) monitors all their messages. Any suspicious messages will result in Wendy placing Alice and Bob into solitary confinement, preventing escape. Alice and Bob must exchange innocuous messages (*overt* channel) containing hidden information (*covert* channel) that Wendy will (ideally) not notice.

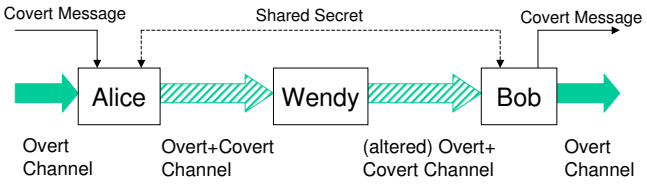

Figure 1. Prisoner problem – communication model for covert channels

If Alice and Bob use two networked computers to communicate, their covert channel may be encoded in subtle variations of the network traffic flowing between their computers during normal, innocuous activity. Alice and Bob must also share a secret – knowledge of how the covert channel is encoded onto the overt channel, and any additional authentication/encryption applied to the covert channel messages.

In practice Alice and Bob may well be networked devices controlled by the same person. For example, a corporate spy might set up Alice and Bob on either sides of the company's network boundary to ex-filtrate restricted information. Wendy is the network manager who can monitor the passing traffic (looking for covert channels) or alter the passing traffic (to disrupt or eliminate covert channels).

Alice and Bob are not required to be the sender and receiver of the overt communication. One or both may be compromised network devices along the overt traffic's path (known as *middlemen*). It is sufficient that Alice can observe and manipulate overt traffic passing through, and Bob can observe the overt+covert traffic passing through.

B. Quake III Arena Network Protocol

Q3's client-server architecture uses UDP/IP to carry information. RFPSCC utilises messages exchanged during game play (Figure 2), and ignores network traffic associated with server-discovery and initial client connection. Clients send *user commands* to the server once per graphics frame rendered (but no more than once every 10 ms). By default the server sends *snapshots* (game world state updates) to clients every 50 ms. Transmissions in each direction are unsynchronised.

During game play user commands indicate player movements, button states (keyboard and mouse) and the currently selected weapon. Figure 3 illustrates the player movements.

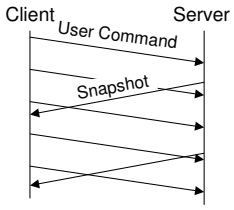

Figure 2. Messages exchanged between Q3 client and server

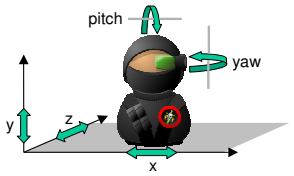

Figure 3. Player character movement

Movement occurs along three axes (x – left/right, y – up/down, z – forward/backward) and change of view angle may be requested along the x- and y-axes (pitch and yaw angles). To compensate for packet loss, each UDP packet sent by the client contains the current and previous user commands.

Snapshots contain the server's authoritative belief about the state of the client's player (position, view angles and player-specific events) as well as the state of all other entities potentially visible to the client's player (positions, view angles and events). Entities can be other human player characters, computer-controlled characters (bots) or objects. Entity state updates are only sent for entities that the client's player can potentially see (which usually is a superset of the entities actually visible on the player's screen). This reduces network traffic and mitigates a source of potential client-side cheating.

Q3 embeds sequence numbers in messages to detect packet loss. Messages sent in 'reliable' mode (such as printing in-game messages on client screens) are retransmitted when lost. Lost user commands and entity state updates are never retransmitted as they are continuously updated anyway. Timestamps are embedded in user commands sent by the client and in player and entity state updates sent by the server. Consequently every update of player state sent by the server can be unambiguously linked to a corresponding (previous) user command sent by a client. Although specific details differ, most FPS game protocols utilise a similar overall design.

Figure 4 illustrates the relationship between player position sent to the server, and player position received by other clients. Define x_i to be client 1's player input for their character's position along an axis (or the view angle along an axis) in user command i and define y_j to be the position or view angle of client 1's character as sent by the server to both clients in snapshot j. We assume x_i and y_i are integer values (or the integer part of real values obtained using the floor operation).

As user commands are usually more frequent than snapshots, each y_j is computed based on the most recently received x_i. Client 2 renders client 1's player on screen based on y_j until it receives y_{j+1} (a period indicated by the boxes).

III. FPS Covert Channel (FPSCC)

A. Communication Model

FPSCC creates a covert channel using traffic between two game clients. Alice (sender) and Bob (receiver) may be deliberately built into actual clients, or be middlemen

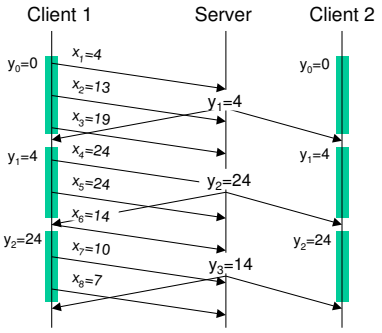

Figure 4. Example user command values and server snapshot values

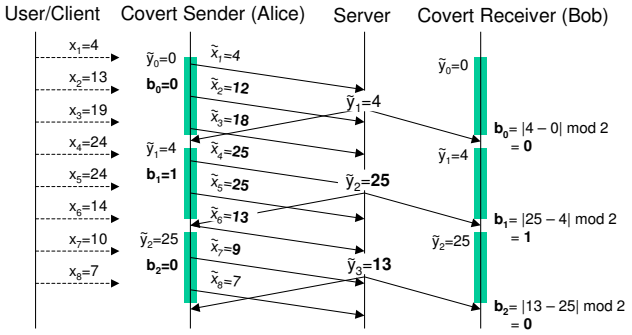

Figure 5. Example of covert channel encoding

manipulating game traffic of unwitting players. Alice encodes covert information by modulating x_i from client 1 (Figure 4) with visually imperceptible fluctuations in character position or view angles. Bob decodes the covert information from y_j updates arriving in successive snapshots.

FPSCC aims to avoid detection by either the players controlling the game clients, or by an adversary (Wendy). We presume that Alice and Bob are not satisfied with using the game's built-in chat function or external messaging or voice communication programs (as these are easily monitored by a suitably-placed Wendy). In previous tests human players did not notice the use of FPSCC and FPSCC-manipulated game traffic looks similar to normal traffic [4]. However, a more in-depth study of the security of the channel remains as future work (many covert channels in the literature are detectable if Wendy has knowledge of their encoding).

B. Encoding and Decoding

FPSCC modulates players view angle commands for pitch (-87 to $+87$ degrees) and yaw (-180 to 180 degrees), as player view angles are (with small exceptions discussed later) almost entirely dictated by player input. We leverage the fact that Q3 encodes more detail in x_i and y_j than can normally be resolved visually by the human player. Other information in user commands is less suitable. Position information may be perturbed by various 'forces' acting on a player's character, making it hard to predict y_j from x_i. Surreptitious manipulation of mouse button or key state is harder to hide from players, and would result in an extremely low throughput.

As x_i conveys relative view angles we use *changes* in view angles to encode covert information. To minimise detection, FPSCC only encodes covert information between two snapshots when players are adjusting their character's view. If players stop moving their view, the covert channel pauses. Our initial experiments suggested that FPSCC is effectively masked if FPSCC-induced changes are small compared to the player's own input [4].

Figure 5 illustrates the encoding of covert bits (using only one angle), with a zero start value ($y_0 = 0$) and the same user input as Figure 4. One bit b_n of covert information is encoded per angle change – an even change signals a 0-bit and an odd change signals a 1-bit. Angles modified by Alice ($\tilde{x}_i$ and $\tilde{y}_j$) are shown in bold, shaded regions indicate the time periods in which a covert bit is transmitted.

Encoding and decoding can be summarised as follows (for details the reader is referred to [4]). Alice encodes N bits of covert information with an integer value of b ($0 \leq b \leq 2^N - 1$) into each angle change so that

$$b = \left|\tilde{y}_j - \tilde{y}_{j-1}\right| \; mod \, 2^N, \tag{1}$$

where $\tilde{y}_j$ and $\tilde{y}_{j-1}$ are the angle values manipulated by Alice. However, Alice can only indirectly modify y_j by modifying the user input x_i, adding a small δ_i:

$$\tilde{y}_j = x_i + \delta_i. \tag{2}$$

From equation 1 and equation 2 follows [4]:

$$\delta_i = \begin{cases} b - \left(x_i - \tilde{y}_{j-1}\right) mod \, 2^N & x_i - \tilde{y}_{j-1} \geq 0 \\ -b - \left(x_i - \tilde{y}_{j-1}\right) mod \, 2^N & x_i - \tilde{y}_{j-1} < 0 \end{cases}. \tag{3}$$

Bob decodes the information using Equation 1, but cannot distinguish between angle changes that do or do not contain covert information. Unless Alice always has covert information to transmit, from time to time Alice must send dummy bits or cease encoding covert bits. In Section IV we introduce an additional layer of encoding so Bob may reliably determine if Alice is actually sending covert data.

In this paper we focus on using FPSCC for *unicast* transmission. Alice knows the in-game identity of Bob's client and only sends covert data when Bob's character is 'visible' to Alice (determined from the snapshots). The covert channel will pause whenever Bob is not visible. If Alice's client's identity (e.g. player name) is known, Bob can focus on decoding $\tilde{y}_j$ updates relating to that specific player. If not, Bob can simply attempt to decode covert information from $\tilde{y}_j$ updates relating to every player. Any stream of $\tilde{y}_j$ updates that generates 'meaningful' covert information identifies Alice. (Meaningful could mean pre-defined bit sequences [2] or previously agreed-upon data structures.)

For Alice to compute the correct δ_i based on $\tilde{y}_{j-1}$ the round trip time (RTT) between client and server, plus the time between two user commands, must be smaller than the time between two snapshots. If Δ_u is the time between user

commands and Δ_s the time between snapshots the maximum tolerable RTT is:

$$RTT \leq \Delta_s - \Delta_u. \quad (4)$$

With snapshots every 50 ms, and user commands often every 10 ms, we are apparently limited to $RTT \leq 40$ ms. A small modification allows FPSCC to work over larger RTTs at reduced throughput. Bits b_n are actually sent in m server snapshots, where $m \geq 2$ and $m \cdot \Delta_s - \Delta_u \geq RTT$. Server-time timestamps t_s are used to control which snapshots contain covert bits. Alice and Bob only encode/decode from snapshots where $t_s \bmod m = 0$.

C. Sources of Bit Errors

FPSCC suffers from bit errors: *substitutions* (bits changed), *deletions* (bits completely lost) and *insertions* (bits inserted).

To cull the number of objects rendered each frame, reduce network traffic and mitigate cheating (chapter 7, [5]), snapshots only send the state of 'potentially visible' entities to a player. Potential visibility is determined from visibility data inside the map file and the actual positions of player and entity on the map [9]. Potential visibility in Q3 is asymmetric (Bob may not receive Alice's state when Alice receives Bob's state), causing bit insertion and deletion errors [4].

Players respawn at various map locations after death or when they enter teleportation devices. Respawning forces a change in the player's position and view angles, which causes substitution errors if the dead (or teleported) player respawns within the visible range of the other player.

If a client requests pitch angles below -87 or above $+87$ degrees the Q3 server clamps the angles advertised in subsequent snapshots. If $\tilde{x}_i$ is outside the allowed range, the corresponding $\tilde{y}_j$ will be clamped to either -87 or $+87$. This causes substitution errors in the snapshot where the angle is clamped and bit deletions in further snapshots (while the angle is clamped). Moving platforms (movers) that player characters can stand on also introduce errors. A rotating mover introduces view angle changes unrelated to the player's input, potentially introducing bit substitutions.

Q3's use of sequence numbers can detect re-ordering of user commands or snapshots (due to UDP/IP packets being lost or re-ordered). However, late user commands or snapshots are simply ignored, and so re-ordered packets are effectively lost.

User commands are redundant, as several are sent between two snapshots. If no user commands reach the server between snapshots, Alice cannot send any bits. However, if some user commands arrive it is crucial that at least one has covert bits encoded based on the angle from the previous snapshot. Otherwise, substitution errors can occur. Lost user commands can never cause deletions or insertions, because Alice always knows whether any bits were sent from the snapshots.

Loss of snapshots is worse than loss of user commands. If the same snapshot is lost for Alice and Bob FPSCC remains unaffected. But a snapshot that is lost for either Alice or Bob causes bit deletions/insertions in the lost snapshot as well as possible substitution errors in the following snapshot.

IV. Reliable FPSCC

Our novel scheme, tailored to the characteristics of FPSCC's overt channel, adds bit synchronisation, a framing layer to provide frame/byte synchronisation and a transport layer to provide encryption.

A. Bit Synchronisation

The basic idea of the scheme is that Alice explicitly lets Bob know whether she can see him or not (and in the same way Bob informs Alice). This requires the use of two special channel symbols. If Alice sends an UNSYNC symbol to Bob, she indicates that she cannot see Bob. If Alice sends a SYNC symbol to Bob, she indicates that she can see Bob. A drawback of using two special symbols is the increased amplitude of the induced angle changes in order to have $2^N + 2$ symbols. (Alice could send special bit patterns instead of using two special symbols, but this would decrease the throughput.)

Our scheme works even in the presence of substitution errors, because Alice always determines whether she has sent (UN)SYNC from the snapshots (rather than what she *intended* to send). Since RFPSCC is used as bi-directional channel we extend our notion of covert sender and receiver to covert peers.

Alice and Bob implement a state machine as shown in Figure 6. Initially Alice and Bob are in IDLE state. A peer in IDLE state sends UNSYNC symbols. When a peer in IDLE state sees the other peer it goes into LISTEN state. In this state a peer sends SYNC symbols. Only when both peers send a SYNC to each other in the same snapshot, the channel's state changes to OPEN (because only then both peers can be sure that they can see each other). Covert data is only exchanged when the channel is OPEN. An OPEN channel goes back into LISTEN state if an UNSYNC or SYNC is received, or back into IDLE state if visibility is lost. The time period the channel is OPEN is called a *transmission period*.

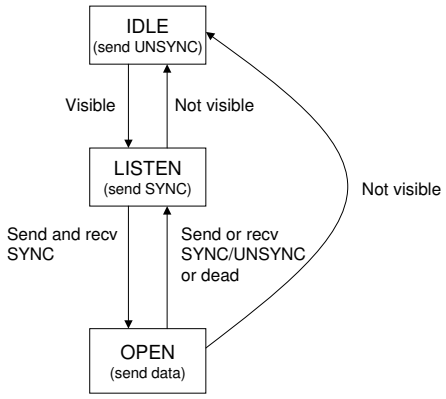

Figure 6. Bit synchronisation state machine

Furthermore, Alice and Bob can only enter OPEN state if both players are alive, and they have to change from OPEN to LISTEN if one or both players die. This is to prevent the case that a dead player loses visibility and is then unable to send an UNSYNC (dead players cannot send anything because they

cannot move). Q3 sends player-death signals synchronously to all players, so no further synchronisation problem arises.

There are two causes of teardown. Firstly, if a peer loses visibility to the other peer, it changes into IDLE state. From the next snapshot on it then sends UNSYNCs. The other peer either loses visibility or receives an UNSYNC and then also ends the transmission period. Secondly, if one or both peers' players die, both peers go into LISTEN state.

In the first case, whichever peer lost visibility last has potentially sent bits the other peer could not receive. To avoid bit deletions these bits need to be re-sent. The problem is to determine the exact number of bits. If a peer receives an UNSYNC symbol it knows that the other peer lost visibility in the previous snapshot and can re-send any bits sent in this and the previous snapshot. (In the rare case that there is no user angle change in the snapshot following the visibility loss, RFPSCC enforces an angle change.)

However, there is still the case that neither peer receives an UNSYNC, making it impossible to re-send the correct number of bits. This happens when one peer loses visibility and the other peer loses visibility in the following snapshot. This problem can only be solved by involving the framing layer (see next subsection). Nevertheless, note that the maximum number of bit deletions is limited to $4N$ (both angles changed in this and the previous snapshot).

RFPSCC also enforces a de-synchronisation of the channel if a peer detects that it did not send the bits intended to be send (as determined from the actual angle values in the next snapshot). This prevents bit errors from movers. To avoid pitch-clamping-related errors, a peer does not send any bits in snapshots where the angle (plus FPSCC's modifications) is clamped. Similarly, a peer does not decode any bits while the angle is clamped.

B. Framing and Transport

The main tasks of the framing layer are to solve the bit synchronisation problem and to identify blocks of bytes in the bit stream (byte synchronisation). There exist a number of framing techniques, for example block length (fixed or variable), or start of frame sequence and bit stuffing (e.g. High-level Data Link Control – HDLC), or Cyclic Redundancy Checksum (CRC) (e.g. Asynchronous Transfer Mode – ATM).

The approach of the RFPSCC framer is to use lower-layer information. All the bits in a transmission period are treated as one frame. Any incomplete bytes at the end of frames are dropped by the receiver and re-sent by the sender. The teardown bit synchronisation problem is solved as follows.

Both peers re-send all bits sent in the last two snapshots before teardown, resulting in at most $4N$ bit insertions but no bit deletions (see Figure 7). For $N \leq 2$ at most one byte is inserted. To prevent byte insertions, each peer sends the parity of the length of the previous frame (odd or even) at the start of the following frame (the decoding is delayed by one frame).

If a transmission period is so short that not all parity bits could be sent, the sender will re-send the parity bits in the next transmission period. Since frames are multiples of bytes

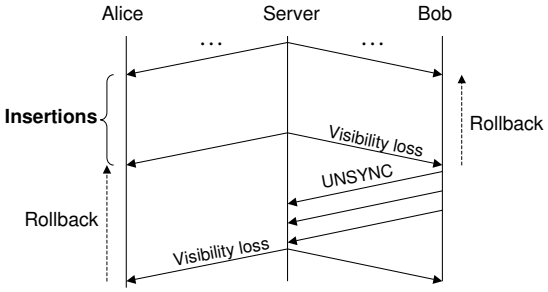

Figure 7. Rollback of bits sent at the end of frames

(byte-aligned) and the maximum number of inserted bits is 8 it is guaranteed that any frames where the sender could not send all parity bits are 'empty' at the receiver. If the actual parity of a frame is different from the parity indicated by the sender, the receiver drops the last byte before passing the data to the transport layer. Our scheme supports larger N by aligning frame sizes on multiples of 8 bits. For example, if frames sizes are aligned to 16 bits (word-aligned) up to 16 inserted bits can be tolerated and hence $N \leq 4$ (frame parity is then computed based on the number of words).

The RFPSCC framing scheme solves the bit synchronisation problem and has low overhead. To ensure byte synchronisation, any bits of incomplete bytes at the end of frames must be dropped at the receiver and re-sent by the sender. Assuming a uniform random distribution of the number of extra bits at the end of frames, the mean overhead is 3.5 bits per frame if frames are byte-aligned or 7.5 bits per frame if frames are word-aligned. Since the number of substitution errors is effectively zero (see Section V), only a single parity bit per frame is needed. In comparison the overhead of HDLC is an 8-bit preamble plus a number of stuffed bits per frame, whereas the overhead for CRC-based framing is 32 bits (ATM).

There are no bit synchronisation errors above the framing layer. This means the transport layer can use a fixed block-size scheme (and therefore block ciphers for encryption). Since there are no substitution errors above the framing-layer further error detection or correction techniques are not necessary.

C. Packet Loss

Only the loss of snapshots affects bit synchronisation. To avoid synchronisation errors we extended our scheme so that transmission periods also end if one or more snapshots were lost that are used for encoding/decoding (depending on the RTT only every m-th snapshot is used).

Each peer can detect lost snapshots by tracking the Q3 sequence numbers. If Alice is sending/receiving data (in state OPEN) and detects that snapshots were lost, she goes into LISTEN state. In the next snapshot she will then send an UNSYNC to Bob who will then also change to LISTEN state. This is the same teardown sequence as described above.

The main difference is that more bits need to be re-sent. To avoid bit deletions in any possible sequence of events Bob needs to re-send all bits sent in the snapshot(s) lost for Alice

and the two snapshots afterwards. However, Alice has no way of indicating to Bob how many snapshots were lost. Hence the number of bits to be re-sent is fixed during RFPSCC operation and must be set according to the maximum possible number of consecutive lost snapshots l_{max} (the maximum loss burst). The maximum number of inserted bits is then $(2+l_{max})\cdot 2N$.

In order to cope with more then 8 inserted bits, frame sizes must be aligned to multiples of 8 bits as described previously. For example, if $N=2$ and bursts of up to 2 lost snapshots shall be supported the maximum number of inserted bits at the end of frames is 16. Word-aligned frames guarantee that all bit insertions can be corrected.

Lost user commands can cause bit substitutions. These are prevented because a peer also ends a transmission period when it detects that the bits that were actually send are not equal to the bits that should have been send (as described earlier).

D. Alternatives

Because of the asymmetric state exchange FPSCC has a significant number of deletions and insertions that occur in bursts. In our experiments we measured 3–4% deletions and 1–2% insertions [4]. In reality rates will likely vary depending on the map, the number and behaviour of players etc.

Some coding schemes have been developed to deal with deletion and insertion channels. However, as Mitzenmacher points out in a recent paper, "[...] the problem of coding for the deletion channel and other channels with synchronization errors [...] remains a largely unstudied area." and "[...] most of the work in this area remains fairly ad hoc." [10]. Many schemes we examined have limitations which make them unsuitable for FPSCC, such as they can handle only small insertion/deletion rates of $< 1\%$, can handle either insertions or deletions but not both, or assume insertions/deletions are not bursty. Also, implementations are usually not readily available.

Furthermore, coding schemes typically require careful parameter tuning in order to provide zero error rates with minimum overhead. But this optimum is difficult to achieve given FPSCC's variable error rates. RFPSCC performs well for varying error rates without any tuning.

V. Achievable Throughput

We evaluated the achievable throughput of RFPSCC depending on various factors such as the number of bits encoded per angle change, network delay and packet loss.

A. Experimental Setup

RFPSCC was prototyped using our Covert Channels Evaluation Framework (CCHEF, software for developing and evaluating covert channels over IP networks) [11]. Our experiments were a mix of tests in a controlled testbed and across real Internet paths, with test machines consisting of two covert game clients, a normal game client and a Q3 game server (all Linux 2.6). The clients ran the Q3 client and CCHEF with FPSCC module. To avoid bias, the covert data was uniform random (same probability of one and zero bits, as one would expect if Alice and Bob encrypted their data).

For delay and loss emulation in the testbed we used the Netem framework built into the Linux kernel [12]. The Linux kernels were recompiled with HZ=1000 in order to emulate delays accurate to ± 1 ms. We verified that Netem's delay and loss emulation is accurate prior to running the experiments.

We emulated RTTs of 25 ms (close server and very fast network access), 75 ms (close server and typical network access) and 125 ms (further away server and typical network access). We chose 125 ms to be the maximum delay as previous research showed that players aim for a maximum RTT of 100–150 ms and higher RTTs noticeably affect their performance [13], [14]. We used constant symmetric delays, as for RFPSCC only the maximum RTT matters.

We emulated loss rates of 0%, 0.5% and 1% (in each direction). For RFPSCC the limiting factor is not the loss rate, but the maximum number of consecutive snapshots lost. We assumed a maximum loss burst of 2 snapshots for 0.5% and 3 snapshots for 1% loss rate (meaning a loss burst lasts less than 150 ms and 200 ms respectively). Previous studies showed that loss in the Internet is usually $\leq 1\%$ [15]. This is consistent with our measurements (see below) showing that loss rates are far below 1% given high-speed broadband access. Furthermore, while Q3 tolerates some loss, the loss rate needs to be reasonably low for good game play [14].

Long measurements with human players are problematic, as exhaustion or change in playing style over time possibly introduces bias in the results. Therefore we used *client-side* bots as players that behave consistently and never get tired (Q3's built-in bots cannot be used as they are part of the server). We configured the bots to play as human-like as possible (e.g. limited their speed). We also performed a limited number of experiments with four human players in order to compare the angle changes per second of bots and humans.

The angle change rates for yaw are surprisingly similar (11.1 changes/s for bots compared to 12.2 changes/s for humans). But for pitch there is a larger difference (4.0 changes/s for bots compared to 9.6 changes/s for humans). On the flat map used in the experiments bots do not need to change pitch very often, but randomness in mouse movements causes more pitch changes for human players. This means that throughput of RFPSCC is likely to be up to 40% higher for human players.

B. Results

We measured the throughput and bit error rate depending on the number of players, bits encoded per angle (bpa), RTT and packet loss rate. For each distinct parameter setting we let the bots play five 1-hour games on the map *q3dm1*. The map was restarted every 10 minutes, because deathmatch games typically run for only 10–15 minutes (whoever has the most kills wins) and then the map is restarted.

The throughput in both directions (Alice to Bob, Bob to Alice) is very similar (differences ≤ 0.1 bits/s) and hence we only plot the means in the following graphs. Figure 8(left) compares the average throughput over increasing RTT for 1 and 2 covert bits encoded per angle change (bpa) and 2 or 3 players at 0% packet loss. Figure 8(right) compares the

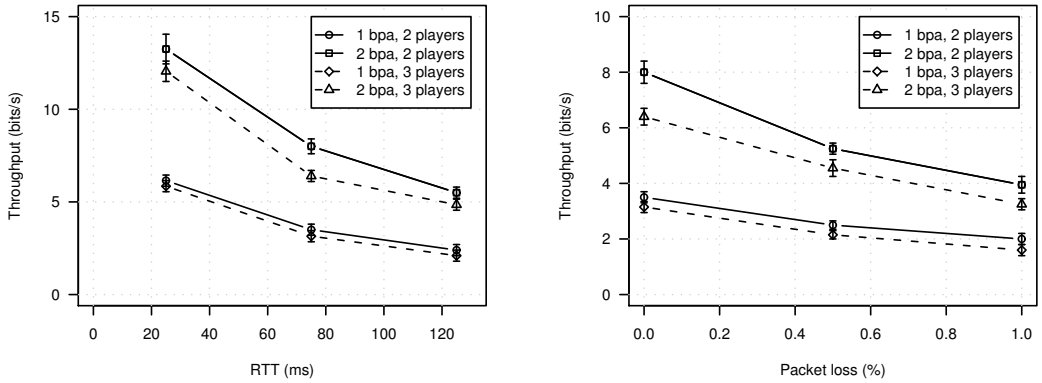

Figure 8. RFPSCC throughput depending on RTT (left) and packet loss rate (right)

average throughput over increasing loss rate for 1 and 2 bpa and 2 or 3 players at an RTT of 75 ms (results for other RTTs omitted for space reasons). The error bars denote the standard deviation. The bit error rate was zero in all experiments.

The throughput of RFPSCC decreases fairly quickly with increasing RTT. However, it decreases slightly less than expected. For example, throughout for 75 ms should be 50% of the throughput for 25 ms (bits encoded in every second snapshot), but the actual throughput reduces only to about 57% (from 6.15 bits/s to 3.5 bits/s). This is because the higher the latency, the longer the bots need to kill each other since aiming becomes more difficult (same effect as for human players [14]). For packet loss the throughput of RFPSCC decreases more sharply between 0% and 0.5% and then the reduction is slower. This is because just enabling loss support causes a throughput reduction (and all experiments with 0% loss were run with loss support disabled).

Because of limited resources we could only test RFPSCC with 2 and 3 bot players. The throughput reduces with increasing number of players and since the bots are unaware and un-supportive of RFPSCC this trend would continue for a larger number of players. However, in reality Alice and Bob (if both are players and covert sender/receiver) could improve throughput by staying in range of each other.

We also tested RFPSCC across the Internet. The two Q3/RFPSCC clients were at the same location as before. The Q3 server was located in a different part of Melbourne. At the time of the experiments the clients were 15 hops away from the server and the average RTT was 48–49 ms (as measured by traceroute and ping). In some initial tests we measured snapshot loss rates between 0.0015% and 0.01% with only single snapshots lost each time (indicating our non-bursty testbed loss settings are justified). Hence we configured RFPSCC for a maximum loss burst of 1 snapshot. We performed three 1-hour measurements with 1 bpa and three 1-hour measurements with 2 bpa. The throughput was 3.15 ± 0.3 bits/s for 1 bpa and 6.25 ± 0.2 bits/s for 2 bpa (with zero bit errors). These results are consistent with the testbed measurements.

We conclude that RFPSCC is reliable. The throughput of RFPSCC is fairly low, but it is still of the same magnitude as the throughput of more sophisticated packet timing covert channels that provide 6–20 bits/s [16]. Since game sessions typically last from tens of minutes up to 1–2 hours [17], the overall amount of data that can be transmitted is substantial.

C. Channel Capacity

We propose an information-theoretic channel model for RFPSCC and compare the theoretic maximum achievable transmission rate with zero bit errors (channel capacity) with our empirically measured throughputs. The output of the RFPSCC channel only depends on the input and the errors, but not on previous inputs. Hence for 1 bpa encoding we model the channel as a cascade of a binary symmetric channel (BSC) and a binary memoryless combined deletion/insertion channel with error rates p_S, p_D and p_I (see Figure 9). The model can be easily extended to 2 bpa encoding since the error probabilities are the same for all symbols.

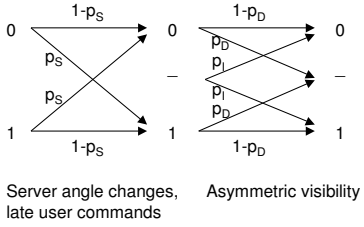

Figure 9. Channel model for RFPSCC with 1 bit per angle encoding

Deletion/insertion/substitution channels have not been very well studied [10]. Exact capacity limits are not known but Gallager proved a lower bound of the capacity [18], [19]

$$C \geq 1 - H(p_D) - H(p_I) - H(p_S), \tag{5}$$

where $H(.)$ is the binary entropy. The Gallager model does not make any assumptions on the error patterns. (The Gallager bound was later improved but given our error rates the difference is marginal.)

From the empirical testbed measurements we examined the average number of deletions, insertions and substitutions (as if no synchronisation mechanism had been used). Insertion and deletion rates are very similar to the rates observed for human players (deletion rates of 3–4% and insertion rates of 1–2% [4]). In the following we assume average rates of $p_D = 0.032$ and $p_I = 0.016$. The sole source of substitution errors were too large time gaps between user commands ($\Delta_u > RTT - \Delta_s$, see Section III-B) since the map has no teleporters or movers and pitch clamping did not occur. For fast clients that send user commands at least every 20 ms the substitution error rate would be zero for our selected RTT values. However, in our case there were few gaps of 30 ms. The substitution error rate increases with increasing RTT, but even for 125 ms it was much smaller than the deletion/insertion rates ($p_S = 0.0031$).

Based on Equation 5 we estimate the lower bound of the capacity in bits/symbol. The number of symbols (angle changes) depends on the RTT and the player behaviour. We computed the average symbol rate f_S for the different RTTs from the experimental data. The maximum achievable transmission rate in bits/s is $R \geq C \cdot f_S$. Table I compares the maximum transmission rates with the measured throughputs at 0% packet loss for different delays (values rounded).

Table I
COMPARISON OF THE CHANNEL CAPACITY LOWER BOUND WITH EMPIRICALLY MEASURED THROUGHPUTS (BPA = BITS PER ANGLE)

RTT (ms)	Maximum rate (bits/s)		Throughput (bits/s)	
	1 bpa	2 bpa	1 bpa	2 bpa
25	6.9	13.8	6.2 (90%)	13.2 (96%)
75	4.2	8.4	3.5 (83%)	7.9 (94%)
125	3.1	6.2	2.4 (77%)	5.5 (89%)

Our RFPSCC scheme achieves 77–90% (1 bpa) and 89–96% (2 bpa) of the capacity lower bound. The overhead is roughly 0.6–0.7 bits/s regardless of the RTT and number of bits encoded per angle. Leigh compares the capacity lower bound with the rate of efficient watermark codes (assuming zero substitution errors) [19]. Given the error rates of RFPSCC the best watermark code provides roughly 0.5 bits/symbol meaning it achieves $\sim 75\%$ of the capacity. This is less than what RFPSCC achieves for non-zero substitution error rates.

VI. CONCLUSIONS AND FUTURE WORK

We have proposed and prototyped Reliable FPSCC (RFPSCC), an enhanced version of FPSCC (covert channels within traffic of online first person shooter games [4]). We have also demonstrated an information-theoretic model to estimate RFPSCC channel's capacity and implemented a proof-of-concept using Quake III Arena. Our solution eliminates the synchronisation and substitution bit errors on the original FPSCC covert channel caused by asymmetric state exchange and packet loss in the overt channel (game traffic).

We analysed RFPSCC in a local testbed with different number of players, network delays and packet loss rates, and across the Internet. Using Quake III Arena, RFPSCC is capable of error-free throughput between 2–13 bits/s, even with modest packet loss in the overt channel. RFPSCC's throughput is similar to that of more sophisticated packet-timing covert channels, and is sufficient to covertly engage in low-speed text messaging or general data transfers. Key advantages of RFPSCC are that it is an indirect channel and cannot be eliminated without eliminating the overt channel (the game).

The application of RFPSCC to non-FPS online games or immersive virtual worlds where player or character movements are regularly propagated to other participants (and their clients) remains future research. We also plan a more detailed evaluation of RFPSCC's performance (more trials with human players and different game/network settings) and hope to improve RFPSCC in the presence of large network delays. Finally, we plan to explore techniques for detecting RFPSCC in regular game traffic and to extend our channel model towards packet loss.

REFERENCES

[1] S. Zander, G. Armitage, P. Branch, "A Survey of Covert Channels and Countermeasures in Computer Network Protocols," *IEEE Communications Surveys and Tutorials*, vol. 9, pp. 44–57, October 2007.

[2] S. J. Murdoch, P. Zielinski, "Covert Channels for Collusion in Online Computer Games," in *6th Information Hiding Workshop*, no. 3200 in Lecture Notes in Computer Science, Springer, May 2004.

[3] J. C. Hernandez-Castro, I. Blasco-Lopez, J. M. Estevez-Tapiador, A. Ribagorda-Garnacho, "Steganography in Games: A General Methodology and its Application to the Game of Go," *Computers & Security*, vol. 25, pp. 64–71, February 2006.

[4] S. Zander, G. Armitage, P. Branch, "Covert Channels in Multiplayer First Person Shooter Online Games," in *33rd Annual IEEE Conference on Local Computer Networks (LCN)*, October 2008.

[5] G. Armitage, M. Claypool, P. Branch, *Networking and Online Games – Understanding and Engineering Multiplayer Internet Games*. John Wiley & Sons, April 2006.

[6] Linden Research, Inc., "Second Life – How Corporations Use Virtual Worlds." http://secondlifegrid.net/slfe/corporations-use-virtual-world.

[7] Id Software, "Quake III Arena." http://www.idsoftware.com.

[8] G. J. Simmons, "The Prisoners' Problem and the Subliminal Channel," in *Advances in Cryptology (CRYPTO)*, pp. 51–67, 1983.

[9] M. Abrash, *Graphics Programming Black Book, Chapter 64*. 2001. http://www.byte.com/abrash/chapters/gpbb64.pdf.

[10] M. Mitzenmacher, "A Survey of Results for Deletion Channels and Related Synchronization Channels." http://www.eecs.harvard.edu/~michaelm/postscripts/DelSurvey.pdf.

[11] S. Zander, "CCHEF – Covert Channels Evaluation Framework," 2007. http://caia.swin.edu.au/cv/szander/cc/cchef/.

[12] Linux Network Developers, "Netem," 2008. http://www.linuxfoundation.org/en/Net:Netem.

[13] G. J. Armitage, "An Experimental Estimation of Latency Sensitivity in Multiplayer Quake3," in *11th IEEE International Conference on Networks (ICON)*, September 2003.

[14] S. Zander, G. Armitage, "Empirically Measuring the QoS Sensitivity of Interactive Online Game Players," in *Australian Telecommunications and Network Applications Conference (ATNAC)*, December 2004.

[15] Y. Zhang, N. Duffield, V. Paxson, S. Shenker, "On the Constancy of Internet Path Properties," in *1st ACM SIGCOMM Internet Measurement Workshop*, 2001.

[16] S. Gianvecchio, H. Wang, D. Wijesekera, S. Jajodia, "Model-Based Covert Timing Channels: Automated Modeling and Evasion," in *Recent Advances in Intrusion Detection (RAID) Symposium*, September 2008.

[17] S. Zander, D. Kennedy, G. Armitage, "Dissecting Server-Discovery Traffic Patterns Generated By Multiplayer First Person Shooter Games," in *Netgames Workshop*, October 2005.

[18] R. G. Gallager, "Sequential Decoding for Binary Channels With Noise and Synchronization Errors," tech. rep., MIT Lincoln Lab, October 1961.

[19] D. Leigh, "Capacity of Insertion and Deletion Channels," tech. rep., July 2001. http://www.inference.phy.cam.ac.uk/is/papers/leigh.ps.