The YS protocol

Vincent A.
vincentweb31@gmail.com

December 26, 2011

The sources of this document can be found on this repository `https://bitbucket.org/vincentweb/ys_proto/src`.

Contents

1 Introduction

When I started hacking with the YS protocol, I knew nothing about sockets, internet protocols, and serialization, but I had great ambitions. The quest was huge for the ignorant knight I was. After years of patience, reading, experimenting, asking questions, sharing code, ... I ended up to get a "little" idea of the YS protocol. But even with this knowledge, the quest will be huge for you.

2 The YS protocol and its links with TCP

In the OSI model, the YS protocol belongs to the application layer, just above TCP. The choice made by Soji Yamakawa of choosing TCP in the transport layer has the following connections :

- more data is sent since the TCP header is quite big compared to other protocols of the transport layer
- if a packet was lost, TCP waits for the server server to send it again although the following packets were successfully received which leads to phenomenons were you see your opponent flying backward during a network play.
- necessity of separating the YS messages since contrary to the UDP protocol, TCP concatenate all the messages to send in its buffer and send them when the buffer is full enough or old enough. That is why all the YS messages start with an integer giving the size of the message. This issue can be avoided by the use of the TCP PUSH flag.
- the OS implementation of the TCP keep-alive is not mandatory, that is why it is done in the application layer.
- you are certain all you messages were received, however the YS protocol force the client to send a copy of the received message to check it received the same thing, which I think is useless. The flaw is that if you don't reply to those messages, the server keep sending it you!

3 Prerequisites

3.1 Sockets

Available in almost all programming language and operating system.

3.2 Serialisation - data representation

3.2.1 Norms

By serialization I mean the process of converting human readable data to its computer format.

All data is sent in little endian, which is the format used by the most common PC processors.

Shorts are coded on 2 octets. Eg 258=0x12 will become 02 : 01 (little endian)

Integers are coded on 4 octets.

Signed numbers are coded using the two complement.

Floats are coded using the IEEE 754 1985 norm

String are coded in ASCII, the null character is put at the end of the string to determine/delineate/demarcate it.

3.2.2 Exemples

C

```c
#include <string.h>

struct info {
        short        var1;
        unsigned int var2;
        float        var3;
        long         var4;
        char         var5[6];
};

// We encode the data
struct info info2send;
info2send.var1 = 42;
info2send.var2 = 55555;
info2send.var3 = 3.14;
info2send.var4 = 7777777;
strcpy(info2send.var5, "hello");
memcpy(buffer, (char *)&myInfo, 24);

// We decode the data
struct info info2recv
memcpy((char *)&info2recv, buffer, 24);
```

Perl

```perl
$buffer = pack("hIfla5",42,55555,3.14,7777777,"hello");
$data = unpack("hIfla5", buffer);
```

PHP

```php
// Returns a string
$buffer = pack("hIfla5",42,55555,3.14,7777777,"hello");

// Returns an array
$data = unpack("hIfla5", buffer);
```

Python

```python
import struct

# pack will return you the string of the serialized information
buffer = struct.pack("hIfl6s",42,55555,3.14,7777777,"hello")

# unpack will return you the tuple of the unserialized information
data = struct.unpack("hIfl6s", buffer)
```

4 The specifications of the YS protocol

4.1 The header

The YS messages have the following shape:

Length (int)	Type (int)	Data

Every YS message start with:

- a length information = data size + type size = data size + 4
- the type of the packet (the purpose of its content)

For example let's decode the following message:

```
18:00:00:00:01:00:00:00:64:6f:69:6e:67:5f:74:65:73:74:73:00:00:00:00:00:7f:db:32:01
```

$18:00:00:00$ = (int 24) is the size of the message
$01:00:00:00$ = (int 1) is the type of the message, 1 means it's a login message
$64:6f:69:6e:67:5f:74:65:73:74:73:00:00:00:00:00:7f:db:32:01$ is the data

Here is a Python piece of code to cope with the previous example.

```
import struct, socket, sys

s = socket.socket()
s.connect(('127.0.0.1', 7915))

# ...
try:
    size = unpack("I",s.recv(4))[0]
    type = unpack("I",s.recv(4))[0]
    data = s.recv(size)
except:
    print "Reception failure."
    sys.exit(-1)

if type == 1:
        (username, ysVersion) = struct.unpack("16sI", data)
        # ...
elif type == 2:
        # ...
```

4.2 The different types

4.2.1 Login (type=1, 0x1)

CHAR[16]	the user name (the 16^e bit is the null character)
INT	the YSFlight version of the client

In the example above we had:
$64 : 6f : 69 : 6e : 67 : 5f : 74 : 65 : 73 : 74 : 73 : 00 : 00 : 00 : 00 : 00 =$ (doing_tests) the username $7f : db : 32 : 01 =$ (20010207) the YSFlight version

4.2.2 Map (type=4, 0x4)

The client must reply the received message.

CHAR[60]	the name of the map

4.2.3 Entity joined (type=5, 0x5)

The client must reply the received message.

INT	type (65537 = ground, 0 = player/pilot)
INT	id
INT	iff
FLOAT	x (initial position in meters)
FLOAT	z (initial altitude in meters)
FLOAT	y (initial position in meters)
FLOAT	rotation1
FLOAT	rotation2
FLOAT	rotation3
CHAR[64]	name of the ground object or name of the aircraft of the player
INT	gro_id
INT	flag
16 OCTETS	unknown
CHAR[56]	second name of the ground object or name of the pilot

The client must answer a packet of type Acknowledge with the data (int 1, int id) in the case of a ground object, (int 0, int id) in the case of a player.

4.2.4 Acknowledgement (type=6, 0x6)

INT	info1
INT	info2

4.2.5 Flight data (type=11, 0xb)

INT	timer which is incremented in an odd way
INT	ID of the pilot flying
SHORT	info1 (5=the lives of the player are coded on 1 char (strength ¡ 256 most of the cases), 3=the lives are coded on a short)
2 OCTETS	unknown (WARNING: these two octets are only present when info1=3)
FLOAT	x position of the aircraft in meters
FLOAT	z altitude of the aircraft in meters (y axis of scenedit)
FLOAT	y position of the aircraft in meters (z axis of scenedit)
SHORT	heading
SHORT	AOA
SHORT	bank
SHORT	xSpeed
SHORT	ySpeed
SHORT	zSpeed
8 OCTETS	unknown
SHORT	fuel
6 OCTETS	unknown
CHAR	spoilerBrake
CHAR	flapsGear
CHAR	afterburnerSmokeTrailsGunfire (convert in binary)
4 OCTETS	unknown
SHORT	gunAmmo
CHAR	rockets
CHAR	unknown
CHAR	AAM
CHAR	AGM
CHAR	bombs
CHAR/SHORT	lives
2 OCTETS	unknown
CHAR	elevator
CHAR	aileron
2 OCTETS	unknown
CHAR	trim

4.2.6 Player left (type=13, 0xd)

INT	ID of the entity who left
4 OCTETS	unknown

The client must answer a packet of type Acknowledge with the data (int 2, int id).

4.2.7 Aircraft list transmission end (type=16, 0x10)

The client must answer with a packet Acknowledge (int 7, int 0)

4.2.8 Keep-alive (type=17, 0x11)

Empty message which must be sent from time to time to avoid being disconnected by the server.

4.2.9 Ground object left (type=19, 0x13)

INT	ID of the entity who left
4 OCTETS	unknown

The client must answer a packet of type Acknowledge with the data (int 3, int id).

4.2.10 Damages (type=22, 0x16)

INT	kind of victim entity (0=the victim is ground object, 1=the victim is a player)
INT	victim ID, (you get the ID of an entity with the messages of type 5)
INT	kind of killer entity (0=the killer is ground object, 1=the killer is a player)
SHORT	power of the damage If you want to kill an object of strength 3 in one shot, this value must be 3.
SHORT	shot (10=missile/rocket hit its target, 11 gun bullet/bomb hit its target, 12 bomb/rocket explosion (not hit directly))
SHORT	weapon (gun=0, aim9=1, AGM=2, bomb500=3, rocket=4, AIM120=6, bomb250=7, bomb500HD=9, AIM9X=10 ; nothing sent for kamikaze kills!)
4 OCTECTS	unknown

The fault is that anybody can send this message, ie the server will allow client X to send a damage message where Y will be the victim and Z the killer.

4.2.11 YSFlight version (type=29, 0x1d)

INT	the YSFlight version of the server

The client must answer with a packet Acknowledge (int 9, int 0)

4.2.12 Missile allowed option (type=31, 0x1f)

INT	missile option (1=missile allowed by the server). This message can be sent to the clients at any moment, allowing a proxy such as YSPS to change options on the fly!

The client must answer with a packet Acknowledge (int 10, int 0)

4.2.13 Chat message (type=32, 0x20)

8 OCTETS	unknown (they are always equal to zero, they are probably kept for a future feature)
CHAR*	chat message

4.2.14 Weather and server options (type=33, 0x21)

INT	(1=day, else it is night)
INT	options (must be converted in binary, blackout enabled = 2nd bit, collisions enables 4th bit, land everywhere enabled = 6th bit)
FLOAT	windX
FLOAT	windZ
FLOAT	windY
FLOAT	visibility (fog)

The client must answer with a packet Acknowledge (int 4, int 0)

4.2.15 User data (type=37, 0x25)

SHORT	kind of user (2=server not flying, 1=client flying, 0=client not flying, 3=server flying)
SHORT	iff
INT	user ID (if flying)
4 OCTETS	unknown
CHAR*	username

4.2.16 Weapon allowed option (type=39, 0x27)

INT	weapon option (1=weapons allowed by the server). This message can be sent to the clients at any moment.

The client must answer with a packet Acknowledge (int 11, int 0)

4.2.17 Show username option (type=41, 0x29)

The client must reply the received message.

INT	the minimal distance to see the player username

4.2.18 Other server options (type=43, 0x2b)

The client must reply the received message.

4 OCTETS	unknown
CHAR*	the option (eg RADARALTI 200m = aircraft below 200m of altitude won't appear in the radar, NOEXAIRVW TRUE = F3 disabled

4.2.19 Aircraft list (type=44, 0x2c)

This message is use to send to the client the list of the aircraft installed on the server. The client must reply the received packet.

1 OCTET	unknown
CHAR	Number of aircraft sent
2 OCTETS	unknown
CHAR[][]	The concatenation of the aircraft identifier installed on the server.

5 Filling the holes

Wireshark

6 Source code

(C#) YSPS v1 (outdated) YSPS v2 (outdated)
(C++) ys_proto
(C++) ys-net-tools (use the library above)
(Python) YSChat (outdated)
(Python) library of the servers list (v2)
(PHP) library of the servers list