Reverse-Engineering the Minecraft protocol

Graham Edgecombe
Minecraft

- Sandbox building (and survival) game, alpha release in May 2009
- Developed by Markus Persson (Notch)
- Mojang Specifications (now Mojang AB)
Minecraft

- 7 million copies sold (10th best selling PC game)
- Written in Java
- Started reverse-engineering the protocol in October 2009 (few months after release), wrote first open-source server
- Wrote widely used API for saved game files (JNBT)
Minecraft

- Map is a giant grid of blocks
- Single- and multi-player modes
- Players can create/destroy the blocks

Later versions added:
- Basic physics (flowing water, explosions, etc.)
- Monsters/fighting
- Items (tools, food, etc.)
- Weather, day/night cycle
- etc.
Tools

- Packet analyzer (Wireshark)
- Java disassembler (javap, JBE)
- Java decompilers (JAD and JODE)
Obfuscation

- Debugging info removed
- JVM is fairly lenient
- Flow control obfuscation
  - new branches leading to invalid code
  - expression always or never evaluates
  - confuses decompilers
- String encryption
- Static member scrambling
- Unused code added
  - e.g. local variables never used, redundant casts
- Various optimisations
Obfuscation

JVM INSTR dup ;
Object obj;
obj;
obj;
printStackTrace();

_L1:
O = minecraftapplet;
new com.mojang.minecraft.e(this);
j = canvas;
b = i1;
c = j1;
I = flag;
if(canvas == null) goto _L3; else goto _L2
Java Serialization

- Used as part of the saved game format
- Some classes do not get renamed (for compatibility between releases)
Minecraft Classic
Authentication Protocol

Client

minecraft.net

Server
Authentication Protocol

- Client and server downloadable by anyone
  - (although proprietary and obfuscated)

- minecraft.net allows secure authentication between servers/clients, without giving away passwords
Authentication Protocol

Server sends a HTTP POST request to http://minecraft.net/heartbeat.jsp every 90s

POST /heartbeat.jsp HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Host: www.minecraft.net
Content-Length: 89

port=25565&users=0&max=64&name=Test&public=false&version=7&salt=-2421305394947607028
Authentication Protocol

if(k1 % 900 == 0) {
    MinecraftServer minecraftserver1 = this;
    Object obj;
    ((Map) (obj = new HashMap())).put("name", minecraftserver1.d);
    ...
    ((Map) (obj)).put("port", Integer.valueOf(minecraftserver1.s));
    ((Map) (obj)).put("salt", minecraftserver1.w);
    ((Map) (obj)).put("version", Byte.valueOf((byte)7));
    obj = a(((Map) (obj)));
    (new d(minecraftserver1, ((String) (obj)))).start();
}
Authentication Protocol

```java
private static String a(Map map) {
    Object obj;
    obj = "";
    for(Iterator iterator = map.keySet().iterator();
        iterator.hasNext();)
    {
        String s1 = (String)iterator.next();
        if(obj != "")
            obj = (new StringBuilder()).append(((String)(obj))).append("&").toString();
        obj = (new StringBuilder()).append(((String)(obj))).append(s1).append("=").append(URLEncoder.encode(map.get(s1).toString(), "UTF-8")).toString();
    }
    return ((String) (obj));
}
```
Authentication Protocol

```
(URLConnection = (HttpURLConnection)((URL) (obj = new
 URL("http://www.minecraft.net/heartbeat.jsp")).
openConnection())).setRequestMethod("POST");

((DataOutputStream) (obj = new DataOutputStream
(URLConnection.getOutputStream())).writeBytes(a);
((DataOutputStream) (obj)).flush();
((DataOutputStream) (obj)).close();

String s = ((BufferedReader) (obj = new BufferedReader(new
InputStreamReader(URLConnection.getInputStream()))).readLine();
MinecraftServer.a.info((new StringBuilder()).append("To
connect directly to this server, surf to: ").append(s).
toString());```
Authentication Protocol

Server receives a URL back to give to players e.g.

http://www.minecraft.net/classic/play/a9a021709992b18501618569ea317cf5
Authentication Protocol

<applet>

... 

<param name="username" value="Ancient"/>
<param name="sessionid" value="-5519425433450728111">
<param name="haspaid" value="true"/>
<param name="server" value="127.0.0.1">
<param name="port" value="25565">
<param name="mppass" value="48aa44d9cf4ea3a8d9d4a42bc65ea5a0">

</applet>
Server authentication code

Upon login:

```java
String s1 = ((String)aobj[1]).trim();
aobj = (String)aobj[2];

if(k.k && !((String)(aobj)).equals(k.j.a(s1))) {
    a("The name wasn't verified by minecraft.net!");
    return;
}
```
Server authentication code

\[
w = (\text{new StringBuilder()}).append(\"\").append(\text{(new Random()).nextLong()}).toString();
\]
\[
j = \text{new i}(w);
\]

```java
public final class i {
    public i(String s) {
        a = s;
    }

    public final String a(String s) {
        this = (\text{new StringBuilder()}).append(a).append(s).toString();
        (s = \text{MessageDigest.getInstance("MD5")}).update(getBytes(), 0, length());
        return (\text{new BigInteger(1, s.digest())}).toString(16);
    }
    ...
```
Authentication Protocol

mppass = md5(salt + username)

Client sends username and mppass to server (it doesn't know the salt)

Server verifies the mppass is correct (it knows the salt and username)
Game Protocol

```
~% netstat -nap | grep java
 tcp  0 30 10.0.0.1:36333 178.79.130.214: 25565 ESTABLISHED 13262/java
~%
```

Runs on top of TCP, uses port 25565 by default
Decomplied packet encoding code

if(obj == Integer.TYPE) {
    a2.d.putInt(((Number)k).intValue());
}

if(obj == java/lang/String) {
    obj = ((String)k).getBytes("UTF-8");
    for(k = 0; k < 64 && k < obj.length; k++)
        a2.i[k] = obj[k];

    for(k = obj.length; k < 64; k++)
        a2.i[k] = 32; // 32 = 0x20 = space character

    a2.d.put(a2.i);
}
Decompiled packet encoding code

```java
if(obj == [B) {
    byte abyte0[];

    // fills unused bytes with zeros:
    if((abyte0 = (byte[])k).length < 1024)
        abyte0 = Arrays.copyOf(abyte0, 1024);

    a2.d.put(abyte0);
}
```
Decompiled packet structure code

```java
static {
    b = new a(new Class[] {
        Byte.TYPE, java/lang/String, java/lang/String, Byte.TYPE
    });
    c = new a(new Class[0]);
    new a(new Class[0]);
    d = new a(new Class[] {
        Short.TYPE, [B, Byte.TYPE
    });
    e = new a(new Class[] {
        Short.TYPE, Short.TYPE, Short.TYPE
    });
    ...
```
byte byte0 = a2.c.get(0);
com.mojang.minecraft.net.a a4;
if((a4 = com.mojang.minecraft.net.a.a[byte0]) == null)
    throw new IOException((new StringBuilder()).append("Bad command: ").append(byte0).toString());

if(a2.c.remaining() < a4.q + 1)
    break; /* Loop/switch isn't completed */
a2.c.get();
Object aobj[] = new Object[a4.s.length];
for(int i8 = 0; i8 < aobj.length; i8++)
    aobj[i8] = a2.a(a4.s[i8]);
Data types summary

integer: byte, short, int, long

floating point: IEEE 754 float and double

byte array (1024 bytes, shorter arrays padded with zeros)

string (64 bytes of UTF-8, shorter strings padded with spaces)
Game Protocol

- single byte opcode
- constant amount of data for each packet

- unusual in that it is symmetric
  - packet structures same in both directions
  - some fields unused
  - some packets unused
Wireshark dump (login packet)

type (0=login), protocol version, username, mppass (last byte unused)
Login response

example of the symmetric structure:
player name -> server name,
mppass -> server motd,
unused byte -> user type
(0=normal, 0x64=admin)
Level data packets

```
0000  03 04 00 1f 8b 08 00 00 00 00 00 00 00 ec d1 41
0010  11 00 31 0c 03 b1 cc f5 53 58 c7 1f 55 61 ec 23
0020  12 02 ef 78 fe 99 03 00 00 00 00 00 00 00 00 00
...
0400  00 00 00 34
```

type (3=level data)

0x0400 = 1024, number of bytes

0x1f8b08 (GZIP header magic)

last byte = experimented to find it is the progress bar for map loading
Decompiled GZIP code

```java
public static byte[] b(InputStream inputstream) {
    byte abyte0[];
    abyte0 = new byte[(inputstream = new DataInputStream(new GZIPInputStream(inputstream))).readInt()];
    inputstream.readFully(abyte0);
    inputstream.close();
    return abyte0;
    inputstream;
    throw new RuntimeException(inputstream);
}
```

4 byte length
followed by 'length' bytes of data
Decompiled GZIP code

byte abyte0[] = com.mojang.minecraft.level.a.b(new ByteArrayInputStream(((com.mojang.minecraft.net.c) (word4)).a.toByteArray())); // this calls method shown on previous slide
...
Level level1;
level1.setData(word6, word7, word0, abyte0);

public void setData(int j, int i1, int j1, byte abyte0[]) {
    width = j;
    height = j1;
    depth = i1;
    blocks = abyte0;
    ...
}

blocks[(i1 * height + j1) * width + j]
data is the level block array!
Level data packets

0000 02

type (2=start of level data)

0000 04 01 00 00 40 01 00

type (4=end of level data)
width = 256 tiles
depth = 64 tiles
height = 256 tiles
Level data packets

// level start (id 2)
((com.mojang.minecraft.net.c) (word4)).c.a(((Level)(null)));
    word4.a = new ByteArrayOutputStream();

// level data (id 3)
short word5 = ((Short)byte14[0]).shortValue();
    byte abyte1[] = (byte[])(byte[])(byte14[1]);
    byte byte16 = ((Byte)byte14[2]).byteValue();
    ((com.mojang.minecraft.net.c) (word4)).c.p.a(byte16);
    ((com.mojang.minecraft.net.c) (word4)).a.write(abyte1, 0, word5);
// level finish (id 4)
((com.mojang.minecraft.net.c) (word4)).a.close();
byte abyte0[] = com.mojang.minecraft.level.a.b(new ByteArrayInputStream(((com.mojang.minecraft.net.c) (word4)).a.toByteArray()));
word4.a = null;
short word6 = ((Short)byte14[0]).shortValue();
short word7 = ((Short)byte14[1]).shortValue();
short word0 = ((Short)byte14[2]).shortValue();
Level level1;
(level1 = new Level()).setNetworkMode(true);
level1.setData(word6, word7, word0, abyte0);
Updating other players

- 20 ticks per second
- small maps - all players notified of all other player's actions
- 128 players max
  - Negative player ids have special meanings:
  - yourself in movement packets (for teleporting)
  - set spawn position in spawn player packet
  - server 'broadcast' message in chat message packet
Server tick code

```java
int i1 = 0x2faf080;  // 50,000,000 nanoseconds (0.05 seconds)
long l2 = System.nanoTime();
do {
    while(System.nanoTime() - l2 > (long)i1) {
        l2 += i1;
        ...
    }
    Thread.sleep(5L);
} while(true);
```

code for updating players, network I/O, authentication, server heartbeat, saving the map to disk, etc.
Updating other players

- Add player
- Move/rotate player
- Remove player

To save bandwidth:
- delta-encoding position and rotation changes
- packets for only moving, only rotating, and moving & rotating
- absolute versions also exist (if client/server get too far out of sync), also for teleporting
Add player client code

```java
if(byte1 >= 0) {
    byte14 += 128;
    word4 -= 22;
    obj4 = new NetworkPlayer(((com.mojang.minecraft.net.c) (obj2)).
cy, byte1, ((String) (obj4)), i7, word4, k8, (float)(byte14 * 360) / 256F, (float)(byte15 * 360) / 256F);
    ((com.mojang.minecraft.net.c) (obj2)).f.put(Byte.valueOf
(byte1), obj4);
    ((com.mojang.minecraft.net.c) (obj2)).c.d.addEntity(((Entity)
(obj4)));
} else {
    ((com.mojang.minecraft.net.c) (obj2)).c.d.setSpawnPos(i7 / 32,
word4 / 32, k8 / 32, (byte14 * 320) / 256);
    ((com.mojang.minecraft.net.c) (obj2)).c.f.moveTo((float)i7 /
32F, (float)word4 / 32F, (float)k8 / 32F, (float)(byte14 * 360) /
256F, (float)(byte15 * 360) / 256F);
}
```
Client position packet

```java
if(y.e) {
    int i4 = (int)(player.x * 32F);

    int k4 = (int)(player.y * 32F);
    int j5 = (int)(player.z * 32F);
    int j6 = (int)((player.yRot * 256F) / 360F) & 0xff;
    int k7 = (int)((player.xRot * 256F) / 360F) & 0xff;
    c1.b.a(com.mojang.minecraft.net.a.i, new Object[] {
        Integer.valueOf(-1), Integer.valueOf(i4), Integer.
        valueOf(k4), Integer.valueOf(j5), Integer.valueOf(j6),
        Integer.valueOf(k7)
    });
}
```
if(k8 == com.mojang.minecraft.net.a.g)
{
    if(((com.mojang.minecraft.net.c)(word4)).c.d != null)
        ((com.mojang.minecraft.net.c)(word4)).c.d.netSetTile(((Short)byte14[0]).shortValue(), ((Short)byte14[1]).shortValue(), ((Short)byte14[2]).shortValue(), ((Byte)byte14[3]).byteValue());
}
Block changes

- Whole map compressed and sent to new clients
  - server uses lowest GZIP compression - still slow

- Solution: copy map, compress in another thread
  - but map can change while this is done

- Queue up block changes, send them all once client has the map
Chat messages

```java
if(k8 == com.mojang.minecraft.net.a.n) {
    String s1 = (String)byte14[1];
    byte byte7 = ((Byte)byte14[0]).byteValue();
    short word2 = word4;
    if(byte7 < 0) {
        ((com.mojang.minecraft.net.c) (word2)).c.w.a((new
            StringBuilder()).append("&e").append(s1).toString());
    } else {
        ((com.mojang.minecraft.net.c) (word2)).f.get(Byte.
            valueOf(byte7));
        ((com.mojang.minecraft.net.c) (word2)).c.w.a(s1);
    }
}
```
Other packets

- Kicking a player with a reason (e.g. bans, login at another computer)
- Change user type (normal/admin) whilst they are logged in
- Other variants of movement packet
- Ping
- Remove player
Limitations of classic protocol

- **Fixed sized strings/byte arrays**
  - wasted bandwidth (zero/space padding)
  - limited lengths (e.g. 64 byte chat message isn't many chars, esp. with multibyte chars from non-Latin languages)

- **Whole map must be sent and kept in client's memory at once**
  - limited map size - realistically 1024x1024x256 is max
  - wastes bandwidth and memory
Minecraft 'SMP'

- 'Infinite' maps
  - map is split into 16x16x256 chunks
  - only chunks near you are loaded

- Fixes limitations in the protocol
  - longer strings/byte arrays possible
  - no padding - saves bandwidth
  - (as of a few weeks ago) encryption

- Many new features
Server Implementation

- OpenCraft
- Java
- Event-driven single-threaded I/O
  - (using Apache MINA library)
- Customizable with Python scripts
  - (using Jython)

- Interesting scripts made by users:
  - Wireworld
  - Capture the flag
  - Basketball
  - Zombies
Open-Source

- **OpenCraft**
  - 20k+ downloads
  - forked and still developed (OpenCraft ACM, CTF)
- **Lightstone (~100 forks on GitHub)**
- **Glowstone, Spout**
  - popular servers based on Lightstone, active development
- **JNBT**
  - library for the saved game format
  - used in many projects
    - servers
    - map editors
    - etc.
Open-Source

● Both projects had several other contributors

● Lessons learned:
  ○ Don't open the code up too early - contributors often shape the design of the code to suit individual patches rather than the overall system

  ○ Saying no to poor code (with tips for improvements) - hard, but necessary to keep good quality

  ○ Also saying no to unwanted features

● Forks - good and bad
Alternative approaches

hMod/Bukkit
● Bukkit API
  ○ consists mainly of interfaces
  ○ some helper code
● CraftBukkit
  ○ the proprietary server software
  ○ Bukkit developers have a tool to:
    ■ analyse bytecode
    ■ modify it (change names, add extra methods, implement their interfaces)

● Plugins
Summary

Techniques:

● Magic numbers (e.g. GZIP in Minecraft)
● Strings
● Mixture of:
  ○ analysis of packet dumps
  ○ analysis of bytecode & decompiled source code
● Guesswork!
Links

OpenCraft:
http://sourceforge.net/projects/opencraft

Lightstone (rewrite for new protocol):
http://github.com/grahamedgecombe/lightstone

Community protocol/file format documentation:
http://wiki.vg/

Slides: