Packet data structure?

Asked 15 years, 11 months ago Modified 15 years, 11 months ago

Viewed 4k times

4

I'm designing a game server and I have never done anything like this before. I was just wondering what a good structure for a packet would be data-wise? I am using TCP if it matters. Here's an example, and what I was considering using as of now:

(each value in brackets is a byte)

```
[Packet length][Action ID][Number of Parameters]
[Parameter 1 data length as int][Parameter 1 data type][Parameter 1 data (multi byte)]
[Parameter 2 data length as int][Parameter 2 data type][Parameter 2 data (multi byte)]
[Parameter n data length as int][Parameter n data type][Parameter n data (multi byte)]
```

Like I said, I really have never done anything like this before so what I have above could be complete bull, which is why I'm asking ;). Also, is passing the total packet length even necessary?

network-programming packet

Share

Improve this question

edited Dec 27, 2008 at 12:02

Follow

asked Dec 27, 2008 at 11:20

ryeguy
66.8k ● 60 ● 201 ● 263

Hey! Could you break the line of bytes with a newline after each bracket? That makes it easier to read. – mstrobl Dec 27, 2008 at 11:22

5 Answers

Sorted by: Highest score (default)

3

Passing the total packet length is a good idea. It might cost two more bytes, but you can peek and wait for the socket to have a full packet ready to sip before receiving. That makes code easier.

Overall, I agree with brazzy, a language supplied serialization mechanism is preferrable over any self-made.

Other than that (I think you are using a C-ish language without serialization), I would put the packet ID as the first data on the packet data structure. IMHO that's some sort of convention because the first data member of a struct is always at position 0 and any struct can be downcast to that, identifying otherwise anonymous data.

Your compiler may or may not produce packed structures, but that way you can allocate a buffer, read the packet in and then either cast the structure depending on the first data member. If you are out of luck and it does not produce packed structures, be sure to have a serialization method for each struct that will construct from the (obviously non-destination) memory.

Endiannes is a factor, particularly on C-like languages. Be sure to make clear that packets are of the same endianness always or that you can identify a different endian based on a signature or something. An odd thing that's very cool: C# and .NET seems to always hold data in little-endian convention when you access them using like discussed in this post here. Found that out when porting such an application to Mono on a SUN. Cool, but if you have that setup you should use the serialization means of C# anyways.

Other than that, your setup looks very okay!

Share Improve this answer Follow

answered Dec 27, 2008 at 11:37

mstrobl
2,391 ● 15 ● 16

3

Start by considering a much simpler basic wrapper: Tag, Length, Value (TLV). Your basic packet will look then like this:

```
[Tag] [Length] [Value]
```

Tag is a packet identifier (like your action ID).

Length is the packet length. You may need this to tell whether you have the full packet. It will also let you figure out how long the value portion is.

Value contains the actual data. The format of this can be anything.

In your case above, the value data contains a further series of TLV structures (parameter type, length, value). You don't actually need to send the number of parameters, as you can work it from the data length and walking the data.

As others have said, I would put the packet ID (Tag) first. Unless you have cross-platform concerns, I would consider wrapping your application's serialised object in a TLV and sending it across the wire like that. If you make a mistake or want to change later, you can always create a new tag with a different structure.

See Wikipedia for more details on [TLV](TLV).

Share Improve this answer Follow

answered Dec 27, 2008 at 12:46

Mat
86.3k ● 35 ● 94 ● 111

3

To avoid reinventing the wheel, any serialization protocol will work for on the wire data (e.g. XML, JSON), and you might consider looking at BEEP for the basic protocol framework.

BEEP is summed up well in its FAQ document as '*kind of a "best hits" album of the tricks used by experienced application protocol designers since the early 80's.*'

Share Improve this answer Follow

edited Dec 27, 2008 at 14:14

answered Dec 27, 2008 at 13:38

frankodwyer
14k ● 9 ● 52 ● 70

2

There's no reason to make something so complicated like that. I see that you have an action ID, so I suppose there would be a fixed number of actions.

For each action, you would define a data structure, and then you would put each one of those values in the structure. To send it over the wire, you just allocate sum(sizeof(struct.i)) bytes for each element in your structure. So your packet would look like this:

```
[action ID][item 1 (sizeof(item 1 bytes)][item 1
(sizeof(item 2 bytes)]...[item n (sizeof(item n
bytes)]
```

The idea is, you already know the size and type of each variable on each side of the connection is, so you don't need to send that information.

For strings, you can just throw 'em in in a null terminated form, and then when you 'know' to look for a string based on your packet type, start reading and looking for a null.

--

Another option would be to use '\r\n' to delineate your variables. That would require some overhead, and you would have to use text, rather then binary values for numbers. But that way you could just use readline to read each variable. Your packets would look like this

```
[action ID]
[item 1 (as text)]
...
[item n (as text)]
```

--

Finally, simply serializing objects and passing them down the wire is a good way to do this too, with the least amount of code to write. Remember that you don't want to prematurely optimize, and that includes network traffic as well. If it turns out you need to squeeze out a little bit more performance later on you can go back and figure out a more efficient mechanism.

And check out google's protocol buffers, which are supposedly an extreemly fast way to serialize data in a platform-neutral way, kind of like a binary XML, but without nested elements. There's also JSON, which is another platform neutral encoding. Using protocol buffers or JSON would mean you wouldn't have to worry about how to specifically encode the messages.

Share Improve this answer Follow

answered Dec 27, 2008 at 11:44
Chad Okere
4,578 ● 1 ● 22 ● 19

The OP stated different types for parameters. What if one data type is to be an array of values? Your solution is sleeker, but disallows for any containment. – mstrobl Dec 27, 2008 at 11:53

0

Do you want the server to support multiple clients written in different languages? If not, it's probably not necessary to specify the structure exactly; instead use whatever facility for serializing data your language offers, simply to reduce the potential for errors.

If you do need the structure to be portable, the above looks OK, though you should specify stuff like endianness and text encoding as well in that case.

Share Improve this answer Follow

answered Dec 27, 2008 at 11:26
Michael Borgwardt
346k ● 80 ● 486 ● 723