1 Conclusion

We have seen several extensions implemented within the example of an IRC Server written in C. Previously, we have been mostly focused on simply examining the differences between the original and modified servers, without passing judgement on whether those differences are benefits or drawbacks. Within this section, we will focus more on the benefits and drawbacks of the implemented extensions. We finish with a brief discussion of whether these extensions are realistically applicable in a real-world piece of software.

1.1 Non-null Pointers

Consider the extension implementing compile-time checking for null pointer dereferences, discussed above in section 3. This extension, while relatively simple in its final form, is actually incredibly useful. When compared with a C implementation of a function that takes pointers as arguments, this extension offers immense upside. The programmer no longer needs to worry about checking every pointer for null dereferences at the beginning of the function.

One additional benefit of this extension is the possibility of guaranteeing that the pointer returned from a function is not null. Again, this would allow the programmer to worry much less about checking that pointers are safely dereferenced. This would be particularly useful in contexts where we are not easily able to restart a program if it aborts due to a null pointer dereference; for instance, a singular server running communication between multiple clients (like the IRC server example discussed throughout this paper) is not easily able to restart if it is forced to abort due to an unexpected null pointer dereference. If this were to happen, all of the data stored on the server (messages, channels, and users, as well as their associated histories) would be lost.

There are no readily apparent drawbacks to using our non-null extension, except for the idea that this extension likely will result in minor slowdowns to code. While not tested using

any sort of reliable benchmark, consider the following toy example containing code utilizing the non-null extension, compared with code not utilizing this extension.

1.2 Asynchronous I/O

Next, we consider the extension implementing nicer facilities for asynchronous I/O than the syntax provided by plain C code. First, we must note that this extension provides only repackaged syntax, not new functionality, when compared with the `epoll` API. The one major benefit provided by this extension, however, is the increase in readability of the code. We no longer must deal directly with the intricacies of the `epoll` API and the concept of *events*. Instead, the extension allows us to focus on the conceptual ideas in the code; that is, we simply want some function task to be run asynchronously, and we don't need to worry about modifying which tasks the `epoll` instance is tracking manually.

One drawback to our asynchronous extension is that we potentially lose some of the finer grains of control provided to us by the raw `epoll` API. While this lack of control could potentially be improved using a more complex extension, we did not consider such an extension within this paper. Additionally, adding complexity to the extension is perhaps antithetical to the idea of extensions in the first place. We wish to abstract away certain aspects of the host language using new syntax, so while we can achieve a more full control using more complexity, this added complexity may become more confusing than simply using the original host language to achieve the same effect.

1.3 Wuffs

Finally, we consider the Wuffs extension that allows for more secure parsing. This extension is a wonderful example of the benefits of using extensible programming languages. We introduce a secure way of parsing using the Wuffs language, leveraging this new tool to write a secure parsing function. The benefits to using this extension are vast, especially when compared with the original C parsing function. The guarantees offered by Wuffs allow us to

be certain that our program has no vulnerabilities to a variety of attacks. Additionally, with this extension, we no longer must worry about manually writing out the code to make the necessary calls to the Wuffs code after it has been compiled into C. Instead, the extension is able to take care of inserting these function calls as necessary.

Even with the added guarantees provided by the Wuffs compiler (requiring more checks than would be required in C code), the Wuffs extension will not have a noticeable slowdown, and could even offer a moderate speed-up in certain cases. For more details, see the benchmark documentation in the Wuffs repository for a comparison of various Wuffs libraries and default C implementations of those same libraries [**4**].

1.4 Applications in Other Software

We have seen the benefits brought by the extensions introduced in this paper. But are extensions a realistic way of writing software for any real-world application? The work within this paper suggests that extensions do indeed provide concrete benefits in real-world applications. We have seen that extensions, particularly those introducing compile-time checking of various language features, are incredibly useful in saving programmers time and energy when writing new software. It is not unreasonable to assume that an extension similar to the non-null extension discussed in this paper would be incredibly useful in a large-scale C project.

Additionally, extensions similar to the Wuffs extension would be incredibly useful in scenarios where we are dealing with safety-critical pieces of code. Any code that deals with possibly malicious inputs would benefit from the guarantees provided by Wuffs. From buffer overflows to integer overflows or underflows, the Wuffs extension guarantees that we are protected from these attacks, allowing programmers to no longer worry about writing more complicated safety checks in C.