CS 461 - Fall 2016 - Technology Review

Matthew Johnson

Abstract

Abstract goes here

CONTENTS

I. Programming Languages

At the highest level, a main component of our software defined network implementation is the programming language it is written in. This is an important decision to make early in our design, as it affects all choices that follow.

A. Options

1) Go: Our first choice is the Go programming language. This is naturally the strongest choice as the rest of the Ciao infrastructure is written in Go. It would require a very strong argument to create a separate networking mode in another language. While technically possible, it would require a great deal of work to make the different pieces compatible with each other.

2) C: Other than interaction with the rest of the Ciao project, C is another natural choice. C has been around for decades, and has the capability to do nearly every computational and networking task. The libraries are extensive and available and the language is fast.

3) Python: Python is a choice here simply because of its ease of use. Python is very expressive and has nice libraries that abstract away the complicated details of software defined networking. The main downfalls of Python, however, are its reduced speed and space efficiency compared to Go and C. A result of writing a cloud orchestrator in Python is exemplified in the extremely complicated and slow Openstack project [1].

B. Goals for use in design

As stated, the choice of programming language will affect all aspects of our design for this project, from code structure to networking libraries and module design. This choice will easily have the largest impact on our project.

C. Criteria being evaluated

Important criteria to consider is the availability of necessary libraries and of the language and its dependencies itself, the inherent speed of the language to be used, the security features the language offers, the concurrency capabilities, and the overall ease of use.

1) Availability: The most available language in terms of libraries and the language itself (regarding its standard libraries) is obviously C because of how ubiquitous it is, how universally available it is, and how extensive its standard libraries are [2]. Close behind C in availability is Python. Python is nearly as available as C is because of how popular it has become in the last ten years [3]. Python has many libraries that provide simple abstractions to networking functionalities.

Of all these languages, Go is the least available as it is the youngest and least popular of the three. Go is not normally available by default on most operating systems and must be installed by the user. Go does, however, have available libraries that make it very easy to implement networking, as is evidenced by the extent to which they are used in the Ciao project currently [4].

The following figure demonstrates the popularity of C, Go, and Python in the United States in the last ten years.

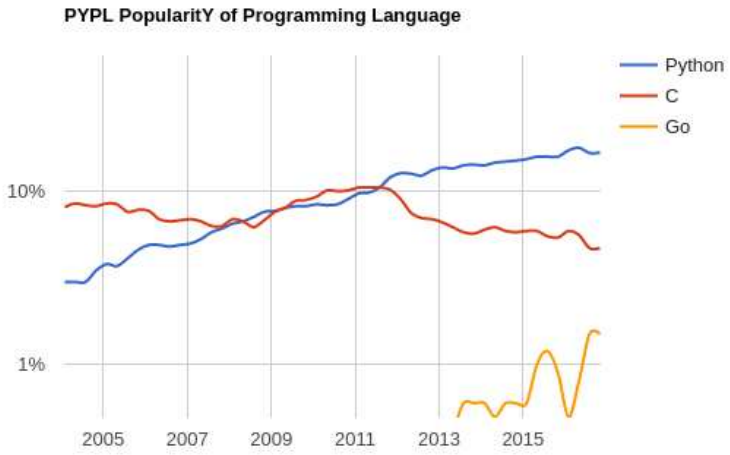

Fig. 1. Python, C, and Go popularity in the US [3]

2) Speed and Space Efficiency: One benefit of lower-level languages like Go and C is how they treat their variables. Go and C treat variables differently than some languages such as Python, which create overhead in order to track type information, and Java, which converts small ints to Integer class instances when placing them in a list. An example of this is in the representations of the same value in Go, Python, and C [5]:

```
var gocon int32 = 2014          // Go:     4 bytes
uint32_t gocon = 2014;          // C:      4 bytes
gocon = 2014                    # Python: 24 bytes
```

Go performs comparably to C with regard to speed, as well [6], which is considerable since C is often the standard for fast programming languages. Compared to Python, as would be expected, Go and C can perform up to 45 times faster depending on the workload [6].

3) Concurrency: Concurrency is a key consideration for programming languages when implementing a software defined network. All operations must happen quickly and in parallel and must scale effortlessly. Therefore, it is necessary that all operations run in their own individual threads.

C has an extensive and established framework for parallel computing by utilizing pthreads. Mutexes can help the programmer protect against race conditions in their code, but the responsibility is up to the programmer to make their software thread-safe. Python has similar tools as C, but the parallelization is handled by a global interpreter lock (GIL). The GIL is a ”mutex that prevents multiple native threads from executing Python bytecodes at once.” GIL is necessary in python because the underlying C code that implements python is not thread-safe. The GIL ”prevents multithreaded CPython programs from taking full advantage of multiprocessor systems in certain situations” [7].

Go, on the other hand, combines the power of C and the ease of use and lock-handling of Python. You can use goroutines (functions that are capable of running concurrently with other functions) to create concurrency. Utilization of goroutines and other builtin language functionalities make concurrency very easy and lightweight in Go. An example from golang-book.com demonstrates how simple and lightweight threads in Go can actually be [8]:

```
package main
```

```
import "fmt"

func f(n int) {
    for i := 0; i < 10; i++ {
        fmt.Println(n, ":", i)
    }
}

func main() {
    go f(0)
    var input string
    fmt.Scanln(&input)
}
```

4) Ease of use: Python is by far the easiest to learn, use, and read. It focuses on "readability, coherence, and software quality" and is recognized by many to be extremely easy to use [9]

C, being the oldest and lowest-level language of the three, is not a simple language to work with. Many things that are normally abstracted away in other languages are required to be programmed explicitly by the programmer. String manipulation is especially difficult in C.

Go is easier to learn than C and makes many improvements in terms of ease of use. It was even designed this way. Go was designed to make programming efficient in large-scale software development across teams with varying levels of experience and skill. It was designed with "built-in concurrency and garbage collection" and includes "rigorous dependency management." [10]. These features are important for the work required by this project.

Another important note is that the rest of Ciao is written in Go, and while possible to create interfaces from C or Python for Go, it would be very difficult. Another option would be to re-implement Ciao in another language, which is so difficult and time consuming that it cannot even be considered as an option. With this consideration, Go is the clear winner in terms of ease of use.

D. Direct Comparison

Language	Availability	Efficiency	Concurrency	Ease of use
Go	3	1	1	1
C	1	2	2	3
Python	2	3	3	2

E. Selection

Based on the criteria explored above, the Go programming language is the language we are selecting to implement our solution in. The remainder of the technical review will be based on the assumption that we will be implementing our solution in Go.

II. Networking Libraries

A. Options

1) :

2) :

3) :

B. Goals for use in design

C. Criteria being evaluated

1) Availability:

2) Speed:

3) Security:

4) Concurrency:

D. Discussion

E. Selection

III. Functional Testing Frameworks

A. Options

1) :

2) :

3) :

B. Goals for use in design

C. Criteria being evaluated

1) Availability:

2) Speed:

3) Security:

4) Concurrency:

D. Discussion

E. Selection

IV. Packet Level Protocols

One of the main components of our system is deciding which protocol will be used to move data from one node to the next. Ciao provides its own data transfer protocol using SSNTP. Comparing how SSNTP works in general to other data transfer formats will provide insight into which protocol should be used.

A. Options

1) SSNTP: Simple and Secure Node Transfer Protocol is Intel's solution for the transfer of data inside of Ciao networks. It is based on Transport Layer Security (TLS), the defacto standard for secure data transfer over the Internet. Part of what sets Ciao apart from the competition is its simplicity, as well as concise message format.[11]

2) TLS: Transport Layer Security, or TLS, is the contemporary standard for secure data transfer across the Internet. It was built as an improvement upon the Secure Socket Layer protocol. It has great appeal as it is widely utilized, modern, and well studied.[12]

3) SSL: Secure Socket Layer, or SSL, was the first version of what eventually became TLS following its third major revision. Originally utilized in Netscape, it wasn't widely utilized until it's third major revision. Secure Socket Layer encrypts at the application layer, allowing broad general use. While being somewhat older than the other protocols on the list, it is well understood and the security has been well proven.[13]

B. Goals for use in design

The primary goal of the selected protocol is fast, stable, secure, and scalable communication between compute nodes and the orchestration node in Ciao. These are simple, but critical goals for the project.

C. Criteria being evaluated

The criteria for each protocol will be their security capabilities, protocol overhead, and ease of use. While stability is a concern, each protocol has shown stability through implementation in other applications many times over. This will be less of a focus for evaluation purposes.

1) Speed: Speed is a major consideration when working with a software defined network. Our final selection for a protocol needs to be fast. Any major delays in communication due to encryption or protocol overhead is a concern. While each implementation of the protocol will have its pros and cons, here the focus will be on application overhead to keep the metric in measurable territory.

SSL has two major components, the protocol, and the handshake protocol. The handshake protocol is important as it ensures the overall security and authenticity of the communication between point A and point B. [14]

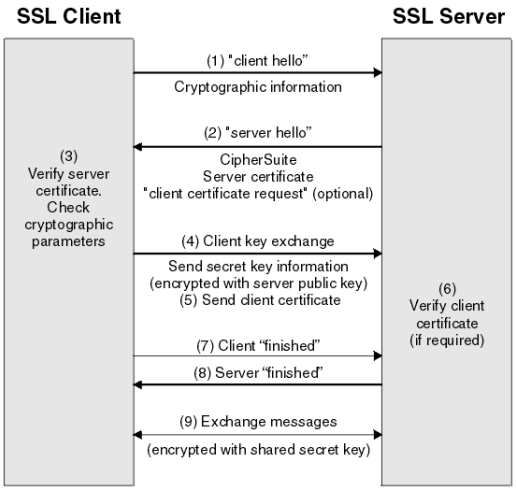

Fig. 2. Simple SSL and TLS Handshake[15]

The SSL/TLS handshakes are identical. As far as overhead the two protocols are very similar. The primary difference between them is their security levels, which will be discussed in the following section.

The SSNTP equivalent of a handshake is called a SSNTP Connection. There is a fundamental difference that needs to be noted between SSNTP and the other two options. The difference is that the computers running SSNTP client software only have to do an authentication handshake, or SNTP Connection in this case, once. Once the client knows which server it is talking to, all connections no longer require a handshake step. This is a massive drop in needed overhead. [11] Once the handshake is established, the client communicates with the server in an asynchronous fashion as needed.

2) Security: Security is important in choosing a communication protocol. If it wasn't, we could just use TCP, the protocol that all of these security protocols encapsulate.

SSL is unfortunately at the bottom of the list as far as the protocols listed here are concerned. Has a few vulnerabilities that have been found in the decade or so. While some of these are less of an issue, one attack in particular called POODLE has caused several major groups to call for SSL 3.0 to be deprecated.[16] While calling into question consideration for using this protocol on this system, it is still better than not having any security or encryption at all.

TLS is much better on this front as it is continually being updated. The latest release, version 1.2, uses AES, RC4, and a few other modern encryption ciphers. [12] Which encryption cipher is used is negotiated between the host and the client before communication begins.

SSNTP uses the TLS encryption protocols. On this front, it is equal to TLS.

3) Accessibility: All of the above protocols are well documented and easy to implement. In the scope of the project, however, SSNTP would be easier to use because other components of the system already use it. For that reason alone, it is simpler to get working for the project.

D. Direct Comparison

Here is a side by side comparison of the protocols in the form of a table:

Protocol	Speed	Security	Accessibility
SSNTP	1	1	1
SSL	2	3	2
TLS	2	1	2

E. Selection

For our project, the technology for protocols will be SSNTP. It does what SSL and TLS do equally security wise, but is much more light weight and is more accessible in the context of the project.

V. nvGRE and VxLAN Switches

A. Options

1) nvGRE:

2) VxLAN:

3) :

B. Goals for use in design

C. Criteria being evaluated

1) Availability:

2) Speed:

3) Security:

4) Concurrency:

D. Discussion

E. Selection

VI. GRE/Linux Bridges

A. Options

1) Linux Bridges:

2) :

3) :

B. Goals for use in design

C. Criteria being evaluated

1) Availability:

2) Speed:

3) Security:

4) Concurrency:

D. Discussion

E. Selection

VII. References

References

[1] J. Bresler, "Tales from the trenches: The good, the bad, and the ugly of openstack operations," *Openstack Superuser*, jan 2015. [Online]. Available: http://superuser.openstack.org/articles/tales-from-the-trenches-the-good-the-bad-and-the-ugly-of-openstack-operations/

[2] tdammers. (2014, dec) What makes c so popular in the age of oop? top answer. [Online]. Available: http://softwareengineering.stackexchange.com/a/141345

[3] P. Carbonnelle. (2016) Pypl popularity of programming languages index. [Online]. Available: http://pypl.github.io/PYPL.html?country=US

[4] M. Castelino. (2016, may) Ciao networking. [Online]. Available: https://github.com/01org/ciao/tree/master/networking

[5] D. Cheney. (2014, jun) Five things that make go fast. [Online]. Available: https://dave.cheney.net/2014/06/07/five-things-that-make-go-fast

[6] Stanford. The computer language benchmarks game. [Online]. Available: http://benchmarksgame.alioth.debian.org/u64q/compare.php

[7] python.org. (2015, jan) Globalinterpreterlock. [Online]. Available: https://wiki.python.org/moin/GlobalInterpreterLock

[8] C. Doxsey. (2016) Concurrency. [Online]. Available: https://www.golang-book.com/books/intro/10

[9] M. Lutz, *Learning Python*, 5th ed. O'Reilly Media, Inc, 2013.

[10] G. Rob Pike. (2012) Go at google: Language design in the service of software engineering. [Online]. Available: https://talks.golang.org/2012/splash.article

[11] I. Corporation. (2016, sep) Simple and secure node transfer protocol. [Online]. Available: https://github.com/01org/ciao/tree/master/ssntp

[12] E. Rescorla. (2008, aug) The transport layer security (tls) protocol version 1.2. [Online]. Available: https://www.ietf.org/rfc/rfc5246.txt

[13] SSL.com. (2016, mar) What is ssl? [Online]. Available: http://info.ssl.com/article.aspx?id=10241

[14] K. W. R. James F. Kurose, *Computer Networking, A Top Down Approach*, 6th ed. Pearson, jan 2013.

[15] IBM. (2016, sep) An overview of the ssl or tls handshake. [Online]. Available: http://www.ibm.com/support/knowledgecenter/SSFKSJ_7.1.0/com.ibm.mq.doc/sy10660_.htm

[16] R. Barnes. (2014, oct) The poodle attack and the end of ssl 3.0. [Online]. Available: https://blog.mozilla.org/security/2014/10/14/the-poodle-attack-and-the-end-of-ssl-3-0/