Uncovering Twilio: Insights into Cloud Communication Services

Ramnatthan Alagappan
ra@cs.wisc.edu

Sourav Das
souravd@cs.wisc.edu

1 Abstract

We study the protocols and architecture of a popular cloud communication service *Twilio* through gray box methods. We develop a simple VoIP service atop *Twilio* for our study. We provide insights into how different components in a cloud communication service interact with each other in various scenarios. We also measure some guarantees with respect to call and message dequeuing provided by Twilio. Our analysis reveals a number of interesting aspects about the Twilio ecosystem and have strong implications for developers who build applications on top of *Twilio* APIs.

2 Introduction

Cloud communication services (CCS) are an upcoming service model which provide sophisticated APIs for enterprises to develop applications, offload communication related tasks from enterprise applications. CCS host switching, telepony infrastruture and storage in the cloud. The services are offered through simple REST APIs for the applications to make use of them. Cloud communication services have a lot of advantages compared to *on-premise* communication infrastructure. Firstly, it takes the burden of communication from the applications and separates it into a separate service. The applications can seamlessly interact with the communication APIs to accomplish complex communication tasks like sending promotional messages to customers, providing critical real-time information to mobile phones etc. Secondly, the development effort involved in integrating an application with CCS is very less compared to developing and maintaining a home-brewed communication infrastructure. Thirdly, enterprises can build communication applications in a cost effective way becasue of the pay-per-use model provided by most the CC services.

Twilio is one of the prominent players in the CCS space.*Twilio* has a huge customer base including popular enterprises like *Coca-Cola*, *WalmartLabs*, *Intuit*, *Box* use *Twilio* to accomplish a wide variety of tasks ranging from two-step authentication, powering lending machines with music, secure file sharing, delivering deals and ads to mobile phones etc. *Twilio* enables lot of new scenarios for businesses that were rather cumbersome to implement in the past. The APIs are simple and intuitive to understand and develop. For our study, we developed a simple VoIP service atop Twilio APIs using a *Twilio* free-account. The entire development and testing took just two developer-weeks. Twilio also provides rich documentation and code examples to accomplish simple things like VoIP communication. We mostly tweaked the code provided in developer forums to get our VoIP service working.

3 Motivation and Related Work

Businesses want to delegate communication from their applications and services because of the advantages provided by CCS. The number of enterprises using CCS has seen a steady increase since its advent. Though there are no enough evidences that this pattern is going to continue, because cloud commucation services provide an attractive cost-model and easy-to-use APIs, we strongly believe that this trend is going to continue in the future. As more and more enterprises start using CCS like *Twilio*, there is a good chance that this traffic may contribute to a good fraction of internet traffic in the future.

There have been lot of studies on VOIP service like Skype in the past [1,2]. There have been recent studies on cloud storage service like *Dropbox* [3]. Best to our knowledge, we are the first to study cloud communication services. Similar studies have been carried out previously - [3] provides a thorough characterization of the Dropbox protocol and traffic patterns. They provide insights into how traffic to *Dropbox* varies across four different networks including home and campus networks. Our study on the other hand does not deal with traffic analysis since we did not have the sophistication of collecting the packet traces in the campus network. Our study aims at studying the architecture and protocols of the entire *Twilio* ecosystem. Our study also aims at finding some possible enhancements to the entire *Twilio* ecosystem. [1] provides a detailed packet level study of the *Skype* protocol. The authors also provide deep insights into the *Skype* architecture. Our study involves studying the architecture of the system and the protocols involved using a suite of gray box tools that we have developed.

The reminder of the paper is organized as follows. In secion 3, we provide a detailed analysis of the *Twilio* ecosystem, architecture of the system, high level protocols for some key scenarios and packet level analysis for two scenarios. We give some detailed measurements with respect to call dequeuing rates in section 4. Then, we present some oddities that we found in the entire ecosystem in section 5. In section 6, we discuss some of the pos-

sible enhancements to the system. Finally we conclude by presenting the implications of our study and our future work.

4 Twilio Ecosystem and Protocol Study

5 Measurements

In this section, we provide some measurements with respect to call and message dequeuing rates. *Twilio* provides guarantees that the places calls and messages will be dequeued at a rate of 1 per sec and placed on to the phones or the browser clients [https://www.twilio.com/help/faq/twilio-basics/what-are-the-limits-on-outbound-calls-and-sms-messages-per-second - Cite this properly]. We measure the degree to which this guarantee is met. All the experiments were conducted on Lenovo W530 laptop with 8 GB RAM running on 4 cores connected to a 30 Mbps internet link.

5.1 Calls

We developed scripts that can queue automated calls to US phones and online browser clients. We queue the first call to the browser client and queue a variable number of calls to a US phone and then finally place one more call to the browser client. *Twilio* deqeues the calls in the order it was placed into the queue. Using *Wireshark* in the browser client, we collect the packet traces and the time stamps of the incoming call connections. We then can calculate the time difference between the first call and the second call received by the browser. A careful reader would identify that the call time stamps may not actual time when the call was dequeued from the queue and it will include the network latency involved in placing the call to the browser client. We noticed that this latency was lesser than 50 ms and so was discounted for the purpose of calculations.

Figure **??** shows the expected dequeue rate, observed dequeue rate and queue rate for calls. The horizontal axis shows increasing number of calls that we place to the phone and the vertical axis shows the queue or dequeue rate. As mentioned before *Twilio* gives a guarantee to dequeue calls at the rate of 1 per second. This is shown by the flat line. We define the queue rate as the rate at which the client library can place calls into the *Twilio* call queue. It is quite possible to achieve higher values of queue rate with high speed links. This value does not necessarily imply that the Twilio server has a throttling mechanism for adding calls into the queue. The *Dequeue rate* line shows the observed dequeue rate in our experiments. We notice that for a small number of calls such as 100, Twilio dequeuing mechanism performs really well than expected. As the number of calls increases, the dequeue rate drop slightly below 1 but stays between 0.9 and 0.95. We believe that this behavior is completely acceptable if further increasing the number of calls to say few thousands. We intend to do this a future work.

texititSummary.*Twilio* call dequeuing guarantees are met and exceeded for small number of calls like 100. The dequeue rate slightly falls below 1 as the number of calls increases from 100. If applications need very critical and real time data to be delivered to phones or clients, we believe it is advisable to use one Twilio number for say around 100 outgoing client/phone connections. For applications that do not need this level of accuracy at which the calls reach the clients, we believe this behavior is completely acceptable.

5.2 Messages

We developed scripts that can queue automated messages to US phones. We queue 100 messages with varying message sizes to a sinlge US phone number. *Twilio* dequeues the messages in FIFO order similar to calls. We noticed that telephone networks take a very variable amount of time to deliver messages to phones. So for calculating the dequeuing rate, we used the timestamp present in the *message resource* in the *Twilio* object store. We query these message resources using the REST APIs and arrive at the dequeue rate by looking at the sent timestamp in the message resource.

Figure **??** shows the expected dequeue rate, observed dequeue rate and queue rate for SMS. The horizontal axis shows increasing size of the messages that we place to the phone and the vertical axis shows the queue or dequeue rate. As mentioned before *Twilio* gives a guarantee to dequeue SMS at the rate of 1 per second. As mentioned earlier in sec-measurements-calls, the queue rate shows the rate at which the client can push messages into the queue. It is interesting to note that the expected rate itself drops as the message size increases. Twilio treats a single message as just 160 bytes. If the message contains say 170 bytes, *Twilio* considers this as two messages and so it can queue at the rate of one message per two seconds and so the expected rate falls to 0.5 from 1. Similarly for a 400 byte message, there are three such chunks and so the expected rate drops to 0.33. It can be noted that the observed rate closely follows the expected rate.

texititSummary.*Twilio* message dequeuing guarantees are well met. An interesting observation is that naive developers who do not notice the size limits of the messages may wrongly believe that *Twilio* can dequeue at a rate of 1 message per second irrespective of the message size. We bring this out clearly in our study.

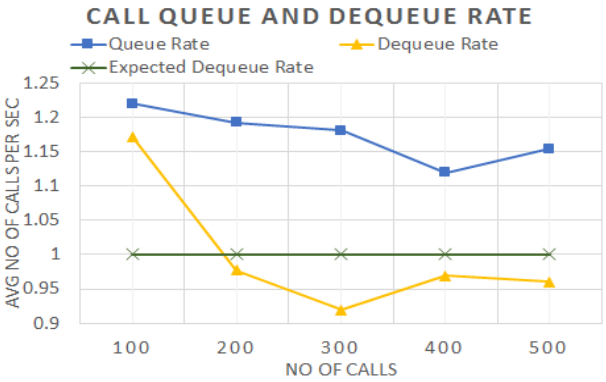

Figure 1: **Call queuing and dequeing.** *The figure shows the queuing and dequeuing rate for the calls*

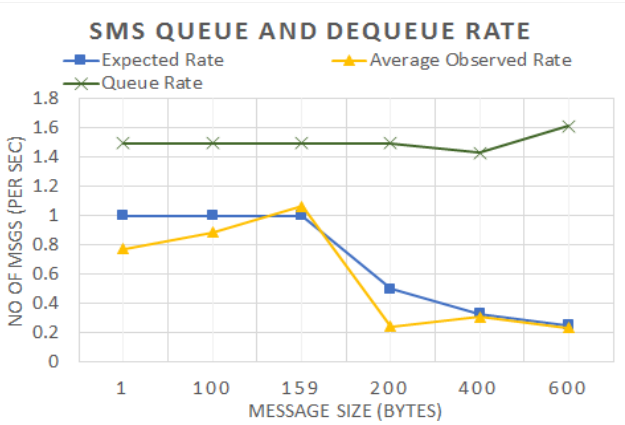

Figure 2: **SMS queuing and dequeing.** *The figure shows the queuing and dequeuing rate for the SMS*

References