



TRACK YOUR TRACK(TYT)

A Capstone project (MINT 709) submitted to the Department of Internetworking (MINT) at the
University of Alberta in Partial Fulfillment of the Requirements
for the Degree of Master of Science.

Shweta Sridharan

Acknowledgments

- “Great things happen to those who don't stop believing, trying, learning, and being grateful.”
- Roy T Bennet

I am going to address the last part of the above-mentioned quote first- being grateful. I have been blessed with an amazing support system, and I cannot be more thankful for that. I'd like to thank my supervisor, **Mr. Juned Noonari**, who has been taking time out of his busy life from another country to help us students with the capstone project. He has been very understanding throughout the course. I'd also like to thank **Dr. Mike McGregor** for being so actively involved in all the student projects despite his busy schedule. I appreciate all the inputs I have gotten from him during the meetings in the course of these two fruitful years. Another person I would like to thank is **Ms. Sharon Gannon**, who has been of tremendous help in coordinating and helping us students with all the registrations, including the capstone project. She has always been available for any qualms we faced during the course of the project. I'd also like to thank **Mr. Shahnawaz** for coordinating and helping us students gain access to the laboratory and the systems on time for all the courses and projects.

I'd like to thank my **fellow MINT students and friends here in Edmonton**, who have helped me tremendously during the course of the project. I would also like to thank my **friends from the North Carolina State University (NCSU)**, USA, who have helped me venture into several new topics during the course of this project. Last but not least, I would like to thank **my parents and my sister** for being the best support system that I could ask for and for continuously motivating me to do better and achieve greater. It is certain that I am in the debt of all the above-mentioned individuals.

TABLE OF CONTENTS

Acknowledgments.....	2
Abstract:	8
Introduction:	9
Background	10
Cloud Computing Services.....	11
Differences in Security Considerations in SaaS, PaaS and IaaS	17
□ Organizations focused on SaaS	17
□ Organizations focused on PaaS	18
□ Organizations focussed on IaaS.....	19
□ For Businesses focussed on SaaS, PaaS, and IaaS	20
CogniCrypt	22
Understanding Cloud Computing.....	27
Case Studies on security breaches:.....	31
Understanding Cloud Security and Data Privacy.....	33
Studying the Cloud Security with an example- Amazon Web Services	34
Valid Accounts- Initial Access	35
Redundant Access- Persistence.....	39

Revert Cloud Instance- Defense Evasion	40
Basic Preventive Measures to Avoid Security Breaches in Cloud.....	43
Well-defined organizational controls related to cloud purchases and use	43
Cloud vendor landscape evaluations.....	43
Update your Bring Your Own Device (BYOD) Policy	43
Have a well thought through and executable emergency plan	44
Literature Review	46
Verifying Cloud Services- Present and the Future	46
Introduction	46
Methods implemented-	48
Discussion:	50
The QUIC Transport Protocol: Design and Internet-Scale Deployment:	53
Introduction:	53
Major advantages of QUIC:	53
Applications of QUIC today:.....	56
Design and Implementation:.....	57
Server CPU Utilization	65
Experiment Conducted:	66
Some Take away points:.....	67

How Secure and quick is QUIC? Provable Security and Performance Analyses.....	68
Introduction:	68
Attacks under consideration:	70
Multi-Hop Packet Tracking for Experimental Facilities	73
Introduction:	73
Architecture:	74
Methodology:	75
Discussion:	76
QUIC.cloud	78
Theory- Track your Track:	83
Introduction:.....	83
What is TLS?	84
What is QUIC?.....	86
Challenges:	115
Why TyT then?.....	116
Conclusion:.....	117
REFERENCES:.....	118

TABLE OF FIGURES

Figure 1 Evolution of Cloud Computing	10
Figure 2 Cloud Services.....	11
Figure 3- Demarcation of Cloud Services with an Analogy (Source: Quora)	15
Figure 4- Separation of Responsibilities of Cloud Services (Source: Quora).....	16
Figure 5 Snapshot of AWS Audit Reporting	18
Figure 6 Snapshot of resource access (Resource Management)	19
Figure 7 Snapshot of API Key management in AWS.....	22
Figure 8 Flowchart- CogniCrypt method for Cloud Services.....	24
Figure 9 Selecting a Service	25
Figure 10 Selecting a cryptographic configurator	25
Figure 11 Snapshot of CloudWatch Metrics in AWS.....	27
Figure 12 Snapshot of selection of Scaling Strategy in AWS	28
Figure 13 Snapshot of CloudWatch Dashboard in AWS.....	29
Figure 14 Snapshot of Resource Groups- AWS	30
Figure 15 List of Attacks on Cloud.....	35
Figure 16 Snapshot of a Resource Policy code-AWS	39
Figure 17 Snapshot of amazon CloudTrail events- AWS.....	42
Figure 18 CSV file of the CloudTrail logs from AWS	42
Figure 19 Taxonomy of Cloud Services Security[27]	47
Figure 20 Proposed Verification framework for service Identities	51
Figure 21 QUIC protocol [3]	53
Figure 22 HTTP request over TCP+TLS (source: Quora).....	55
Figure 23 HTTP request over QUIC Protocol (source: Quora)	55
Figure 24 Timeline of QUIC's initial 1-RTT handshake.....	57
Figure 25 Structure of a QUIC packet	60
Figure 26 Comparison of handshake latency[3]	64
Figure 27 Comparison between TCP and QUIC[3].....	65
Figure 28 Unreachability of packet payloads	67

Figure 29 Multi-hop packet analysis architecture.....	74
Figure 30 Snapshot of the Demonstrator	76
Figure 31 Snapshot of the Domain created on QUIC.cloud	81
Figure 32 Snapshot of the Anti-DDOS features of the QUIC.cloud	82
Figure 33 Graph- Cloud CDN Throughput.....	83
Figure 34 Structure of TLS	84
Figure 35 Source- Google Developers Live- QUIC (Slide 14)	87
Figure 36 Forward Error Correction	88
Figure 37 Comparison of latencies	89
Figure 38 (Source: Blackhat)	90
Figure 39 snapshot of the import network logs- quic	93
Figure 40 Snapshot of the QUIC options.....	94
Figure 41 Snapshot of the alternate Service Mappings	95
Figure 42 Event Selection- QUIC network logs	96
Figure 43 QUIC Active Connections log.....	96
Figure 44 Transport Socket Pool logs.....	100
Figure 45 Using TLS for QUIC	100
Figure 46 Interactions between QUIC and TLS	101
Figure 47 Packet protection in QUIC	107

Abstract:

Today, Cloud Computing is completely taking over businesses by delivering high-quality computer services and solutions over several servers, databases, storage spaces, and various software. It provides the required flexibility, agility, and scalability to expand and innovate. All this comes with a cost- Security. In this project, we're going to discuss a method to improve security in the transport level of the network. QUIC (Quick UDP Connections) protocol is a fairly new concept which has several advantages and its share of shortcomings. While there is no concept of Ack messages once the connection has been established, there are high chances of DDoS (Distributed Denial of Service) attacks or data breaches. The reason why we are considering a new protocol like QUIC is mainly because it is easier and more effective to experiment with a protocol under research rather than using a standardized protocol like TCP (Transmission Control Protocol). According to the IETF, Transport Layer Security (TLS) is used to secure QUIC. Once the TLS handshake chunk of data has been delivered to QUIC, it is QUIC's responsibility to deliver it reliably. Each block of data that is produced by TLS is correspondingly associated with the bunch of keys that TLS is presently using. If QUIC requires to retransmit that data, it must use the same keys even if TLS has already updated to newer keys. Additionally, frames associated with transferring data can appear only in the 0-RTT and 1-RTT encryption levels. With this project, I'd like to propose a tracking system to ensure more security and reliability by introducing a tracking factor along with the currently used TLS security architecture on QUIC. According to MA.TTIAS.BE, An eminent characteristic of QUIC is FEC or **Forward Error Correction**. Every packet that gets delivered also includes a part data of the other following packets so that when a packet goes missing, it can be reconstructed without having to retransmit it. Due to this, there is a trade-off: each UDP packet contains more *payload* than is strictly necessary. With this tracking feature, I'd also like to reduce the rate at which packets get retransmitted in case it fails to reach the destination. The underlying idea is to dispense more control to the sender of the packet in order for the sender to monitor the route of the sent packet and detect any breach or interruptions along the way.

Introduction:

Cloud Computing is completely revolutionizing the way businesses today procure computer services. Everything is available from anywhere at the snap of the finger. A "Cloud" refers to a set of hosted resources delivered to a user through software. Cloud computing infrastructures, along with all the data being processed, are dynamic, scalable, and portable. Cloud security controls are required to respond to the environmental variables and accompany workloads and data while at rest and in transit, either as inherent parts of the workloads (e.g., encryption) or dynamically through an Identity and cloud management system and APIs. This aids in protecting the cloud environments from system corruption and data loss. In fact, with cloud services such as AWS (Amazon Web Services), we can host serverless websites today.

Well, the major stranglehold for this vast fast-growing technology is a **security threat**. When you transform to the cloud, you're entrusting a major portion of your business (namely, the organization's intellectual property) to a third party (namely, the cloud service provider). Even the most minute of an opening can create unauthorized access, theft, and tremendous financial loss to the organization (A small calculation has been given as an example in the Literature Review- Verifying Cloud Services- The present and the Future). Cloud is like a reservoir of data today, which means illegal access to this reservoir can lead to data breaches and intrusion of privacy. Hence, securing this data takes precedence.

In this proposed system, we have a tracking system for the TLS security over QUIC protocol used by google chrome and now slowly taking over the cloud platform as well. With this project, I'd like to make the networking overcloud a lot more secure with lesser payload. The idea is for the client to exercise more control over the packets it sends to the server. The ability to track sent packets helps in curbing attacks and removes the need for acknowledgments from the receiver, thereby increasing the data rate. This entire report is divided into two parts- **Understanding Cloud Security today** and the **security design proposal** at a protocol level. I saw this interesting picture from an article- The cloud is here- embrace the transition- How organizations can stop worrying and learn to "think cloud" by Deloitte depicting the cloud services today:

Background

With every passing decade, we end up discussing a new technology, its pros, cons, challenges, and applications. Well, one such technology is Cloud Computing- that has completely changed the way businesses operate today. It has almost become a metaphor for the internet. Ever wondered how the cloud came into being in the first place?

An article in the MIT Technology Review has a picture of George Favaloro in 1996, a Compaq marketing executive holding the Compaq business plan that is known to have been the first use of the term ‘Cloud Computing.’ George and his colleague, a young technologist, Sean O’Sullivan, envisioned all the business software to be moved to the Web, which they named as ‘Cloud computing-enabled applications.’

Over the coming years, the use of cloud computing started spreading rapidly since it contributed to a historic shift in the IT industry in terms of more computer memory, processing power, and applications hosted in remote data centers or what is called as the ‘cloud.’

According to the Great Cloud Migration, Wordpress, the following timeline approximately defines the evolution of cloud computing over time:

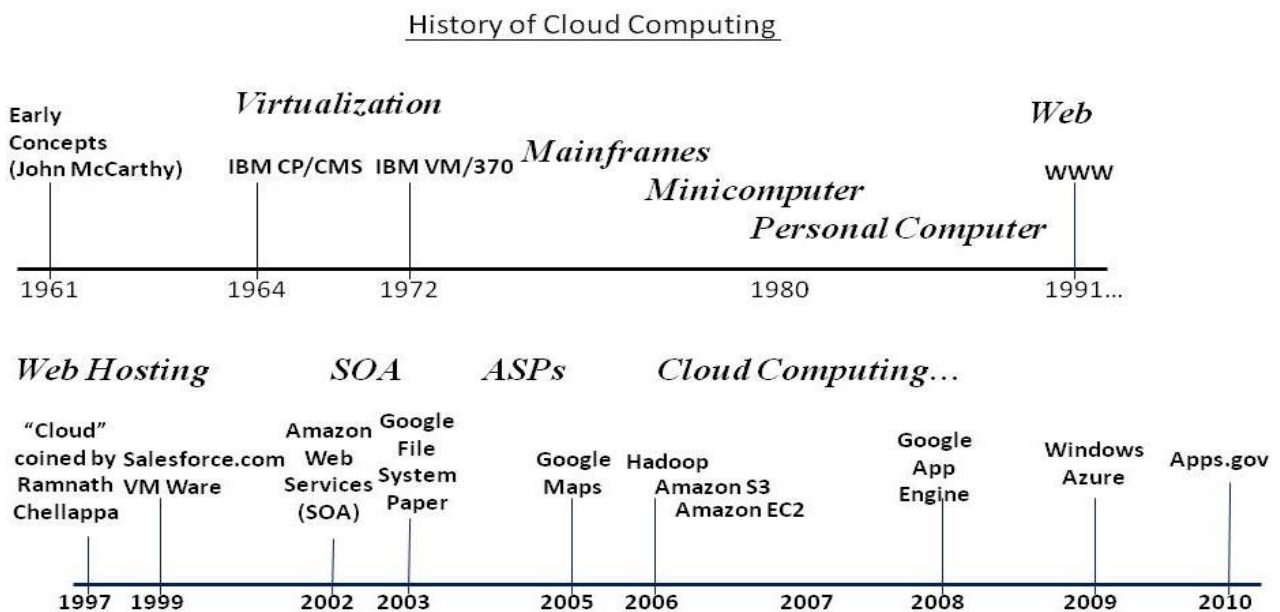


Figure 1 Evolution of Cloud Computing

Cloud Computing Services

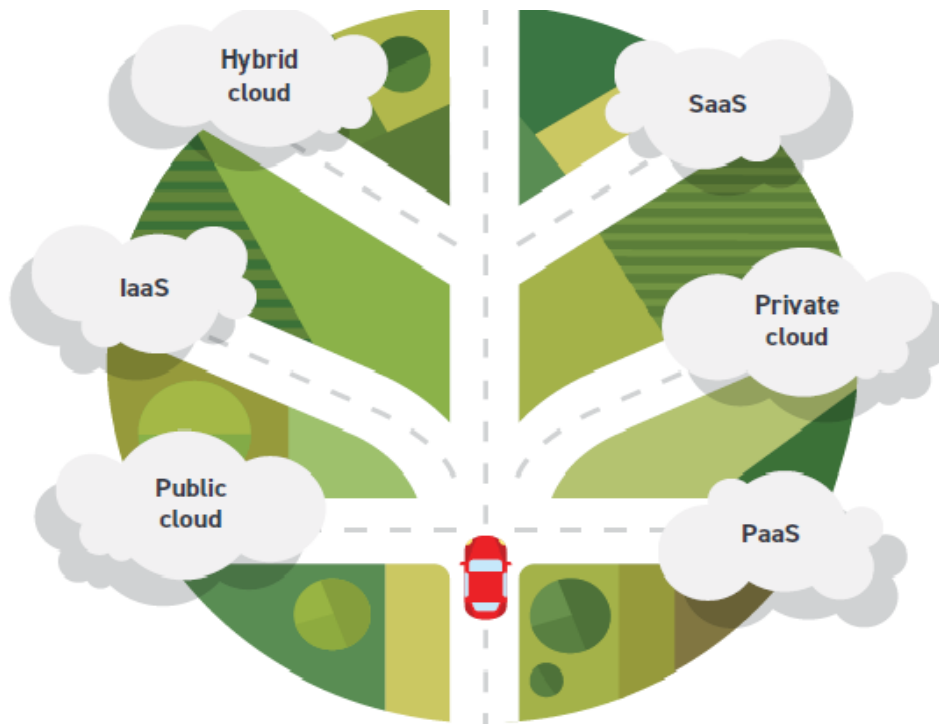


Figure 2 Cloud Services

Let us see what each of these entails:

Public Cloud- It basically means renting cloud services, i.e., you wouldn't have access to the hardware that runs your cloud services. It just stores information and resources that can be used by the public. For example, we all use google's free services like Gmail, Google docs, and google drive. So, where do you possibly think the data is stored, or the website is hosted? All the content and the websites are stored in the cloud, which basically means it is stored in a computer/server elsewhere in the world where they have a data center. The public cloud strives to allow users to share resources whilst maintaining the privacy of their data. Public cloud architecture is entirely virtualized, thereby providing an environment wherein shared resources are leveraged as needed.

Major advantages of Public Cloud:

- An important advantage of public cloud architecture is the **ubiquity**- the ability to access a service or an application on any device that is connected to the internet.
- The next advantage, as I see it is **Simplicity**. The device by itself performs little to no computation,

thereby enabling individuals to conveniently deploy highly complex applications almost anywhere and anytime. Therefore, **no complex set up** and technically skilled (In Cloud Technology) engineers are needed.

- Public cloud is typically designed with **built-in redundancies** in order to prevent data loss. A service provider may store replicated files across several data centers to ensure that the disaster recovery is smooth and fast.
- Finally, Public Cloud is all about unlimited **scalability**. This is precisely because the Public cloud storage providers aid organizations in offloading their physical hardware and the corresponding costs, including server maintenance, power, and cooling technology.

Disadvantages of using Public Cloud:

- The first one that needs to be addressed is **Security**. Your servers are probably in a whole other country governed by a different entity with an entirely different set of security and privacy rules and regulations. It is indeed a total loss of control for the organization.
- The second cause of concern would be **Performance issues**. Your data transmission might be affected by spikes in use across the internet. If applications that are deployed are all heavy duty with large amounts of confidential data, then the public cloud is not the most suitable option.
- The next one is- **Lack of customized options**. In regard to the services provided, since the public cloud providers cater to a large number of customers, the services are rather generic and not very application-specific.

Private Cloud- It basically means owning the cloud services for yourself. You will ideally have complete access to the hardware that runs your services (data center) and will not have to share the resources with any other organization. It is also called the single-tenant environment. It can be created both on and off-premises.

Major advantages of Private Cloud:

- Overcoming the very first disadvantage of the Public Cloud- Better Security. This is certainly because you have complete control over your resources (data center) and can thereby employ your application-tailored security and privacy policies and regulations to it.
- Secondly, we talk about a much better performance. Having the data center in your vicinity

removes any kind of inconvenience caused due to an internet connection. Also, applications of any size and capability can be easily deployed, provided the organization can afford the costs of the servers associated with it.

- The next one in line- More options to customize according to our requirements, size of the company, affordability, and the area available. The services can be tailored specifically, and options can be added or deleted according to the need of the hour.

The disadvantages of using a private cloud:

- **Affordability.** Only big organizations with large chunks of data and big applications in the pipeline to be deployed can afford private cloud services. This is because the hardware costs (Setting up the data center) are quite hefty And require regular maintenance and updates, all of which might prove cumbersome to smaller organizations.
- The second point would be- the need to employ system security engineers and data center technicians in order to operate and regulate the private cloud services. This, in turn, leads to **higher usage of resources** and costs.

Hybrid Cloud- Hybrid cloud is basically a cloud computing environment that uses an amalgamation of private cloud, public cloud, on-premises, and third-party services with an orchestration between the two platforms. It allows workloads to move to and fro between the private and public clouds as computing needs and costs keep changing. When establishing a hybrid cloud, an organization would require the following- A public Infrastructure as Service (IaaS) platform, such as Amazon Web Services (AWS), Microsoft Azure or a Google Cloud Platform; The construction of the private cloud services through a hosted private cloud provider or on-premises and the corresponding Wide Area Network (WAN) connectivity between those two environments (IaaS and Private Cloud Services).

Usually, an enterprise will choose a public cloud to access compute instances, storage resources, or any other specific services, such as serverless computing capabilities or big data analytics. However, an enterprise has no control over the public cloud's architecture, so, for a hybrid cloud deployment, the organization must architect its private cloud to achieve complete compatibility with the desired public

cloud providers. This involves the implementation of a suitable hardware within the data center, including storage, a local area network (LAN), load balancers, and servers.

An enterprise must then deploy a hypervisor or a virtualization layer to create and support virtual machines (VMs) and, in some cases, even containers. Thereafter, the IT teams have the responsibility to install and employ a private cloud software layer, for example, the OpenStack, on top of the hypervisor to deliver all the required cloud capabilities, such as automation and orchestration, self-service, reliability and resilience, chargeback and billing. A private cloud architect will create a list of all the local services available, such as database instances or compute instances, from which the users can choose the service they wish to avail.

The underlying key to create a successful hybrid cloud is to select the cloud software layers and a hypervisor, both of which are entirely compatible with the required/subscribed public cloud services, ensuring proper interoperability with that public cloud's services and application programming interfaces (APIs). The implementation of compatible software and services also aids cloud instances to migrate seamlessly between private and public clouds. A developer can also create advanced applications using a mix of services and resources across the public and private platforms (Software Defined Networking).

Major advantages of using Hybrid cloud:

- You have complete control over the entire scale of architecture, which further allows the organization to customize, create, and deploy services as needed. This also means more flexibility and escalating possibilities to scale.
- Better Security. Since the operations happen right under your nose, security policies can be formulated accordingly.
- Better Speeds. It is much faster to operate even with large applications with large chunks of data.

Major disadvantages are:

- Organizations incorporating Hybrid cloud services usually incur very large costs of operation owing to the entire set up of the hardware and software services.
- Regular Maintenance is required. With such large scale deployment comes tremendous needs for technicians and engineers to regulate and maintain the services. This will, in turn, increase the

costs to the company.

SaaS- Software as A Service- Refers to a software application/service hosted by the service provider on their server to be accessed by the users across the internet without facing the hassle of installing it locally in their respective computers or the intranet.

PaaS- Platform as A Service- Refers to a platform that is typically provided with the OS, server software, server hardware, and network infrastructure, allowing the users to easily develop and deploy applications.

IaaS- Infrastructure as a Service- refers to a service wherein you install the OS, software stack, select the servers (physical or virtual) and then deploy your applications.

On that note, in order to understand the concepts of SaaS, PaaS, and IaaS, I found two very interesting images to understand this concept entirely:

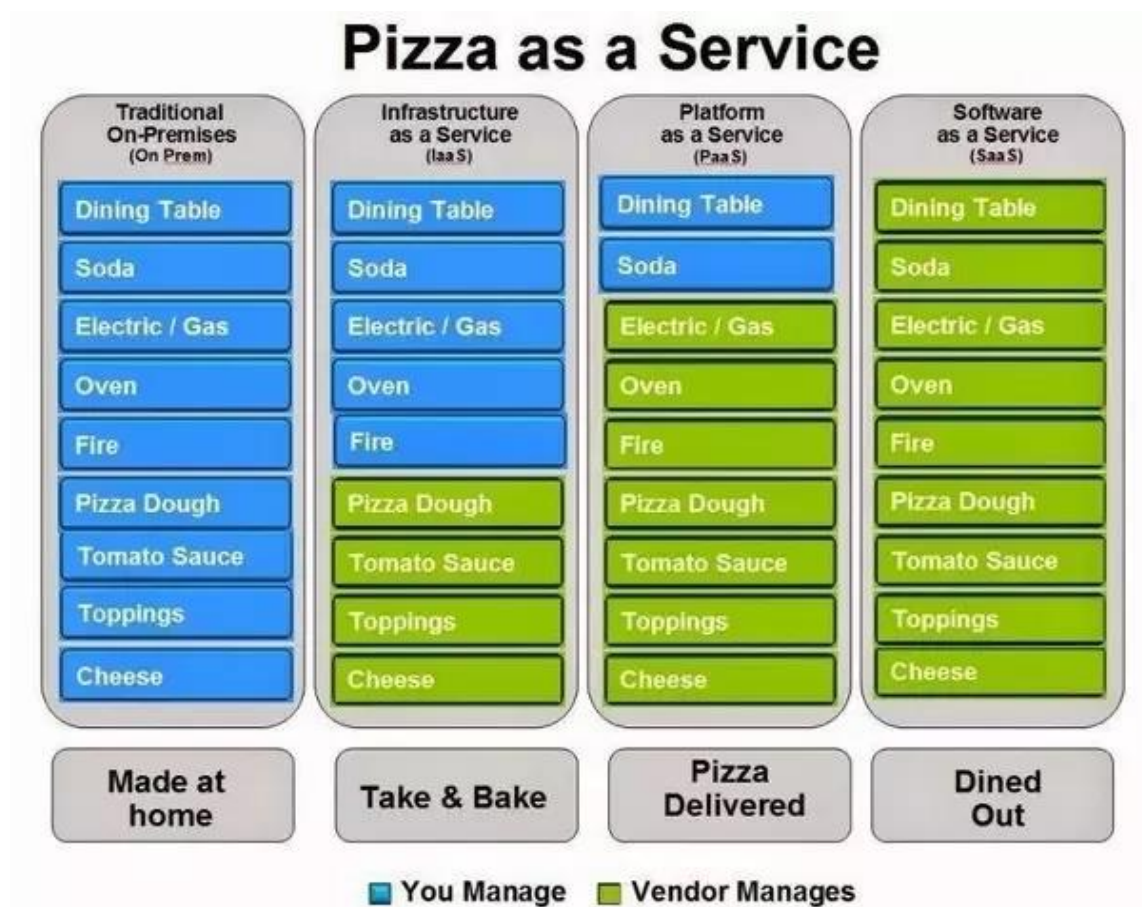


Figure 3- Demarcation of Cloud Services with an Analogy (Source: Quora)

Separation of Responsibilities

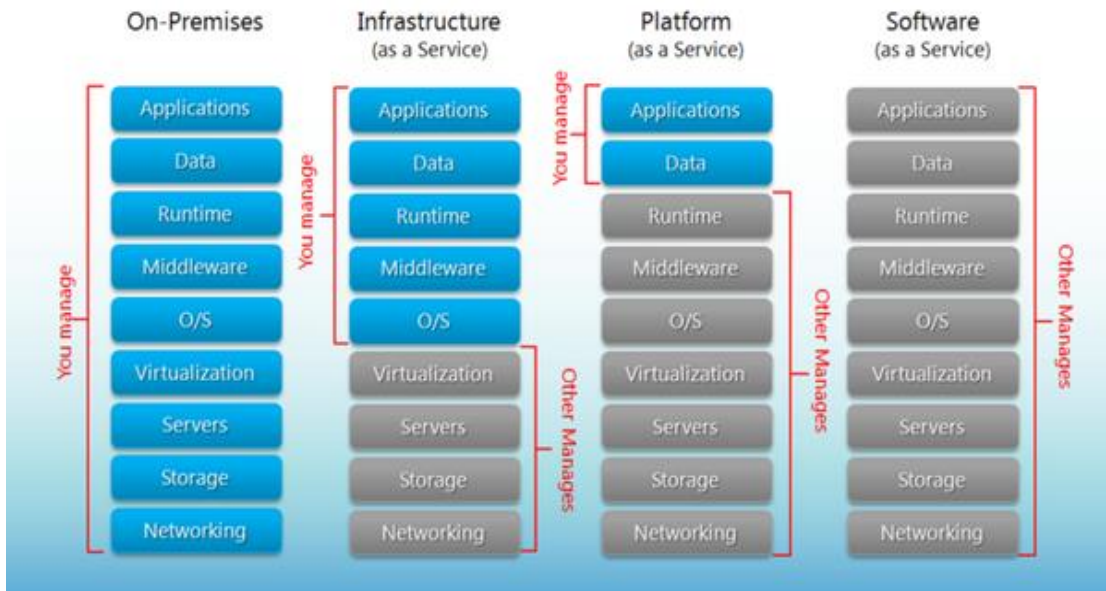


Figure 4- Separation of Responsibilities of Cloud Services (Source: Quora)

- (Figure 2) In case you're preparing pizza at home, it means you have an on-premise environment where you manage everything, i.e., you cook your own cheese, toppings, pizza dough, you own an oven, gas, etc. and you make the entire thing on your own and eat in the comfort of your home. You control how good or bad your pizza is. This is the concept of **On-premises services**.
- In the case of **IaaS**, you purchase the raw materials (i.e., your compute environment, storage disks, your OS, etc.) from your Cloud Service Provider. The infrastructure would be provided to you by the Cloud Provider, and you would not have complete control over it (You will not know where exactly your server is, where your disks are, etc.). But you control about how patching is done on your OS, what workloads you have on your environment etc.
- In **PaaS**, you buy a pizza from a restaurant and eat it at home - i.e., you don't worry about controlling your databases, load balancers, etc. are configured. They are given to you as managed services by your cloud service provider. Various major tasks such as backing up databases etc. become the service providers' responsibility, and you can completely focus on building and deploying your application on the Cloud. The amount of resources that are controlled by an organization reduces one step further.
- In the **SaaS** model, the majority of the services are managed by your service provider, and the

amount of configuration or set up required at the organization's end becomes minimal. The amount of control an organization has on its cloud resources keeps decreasing as we go towards the right in the diagram (Highest control on IaaS to least control on SaaS)

Differences in Security Considerations in SaaS, PaaS and IaaS

- **Organizations focused on SaaS**

The main consideration with relying on Software as a Service would be **not to replicate the organization as such on the cloud**. Large organizations using Cloud services usually face a dilemma. If they potentially have say, ten thousands of employees using Cloud services, is it a good idea to create that many redundant users on the Cloud platform? The solution to avoid this requirement by providing a single sign-on between Cloud (SaaS) and on-premises systems negates this requirement.

However, several users with multiple passwords also prove as a potential security threat to the IT Help Desk resources. The costs and risks associated with multiple passwords are especially burdening for any large organization making its first attempt into incorporating Cloud Computing and leveraging applications or SaaS. For example, if an organization has 20,000 employees, how expensive do you think it is to have the IT department assign new passwords to access Cloud Services for each individual user? For instance, say, the user forgets their password for the SaaS service and eventually resets it; the IT department now has an extra password to take care of.

By incorporating single sign-on capabilities, an organization can ideally enable a user to access both the user's desktops and the Cloud Services via a single password. There are significant economic benefits to this approach in addition to preventing security issues. For instance, single sign-on users are less likely to lose passwords, thereby condensing the assistance required by IT helpdesks. Single sign-on is also very useful for the provisioning and de-provisioning of passwords. If a new employee joins or leaves the organization, there is only a single password to activate or deactivate as opposed to having several passwords to deal with. Therefore, the danger of not having a single sign-on in place for the Cloud is an escalated exposure to security risks (Organization being replicated on the cloud) and the potential for escalated IT Help Desk costs, as

well as the danger of dangling accounts after certain users leave the organizations, which are then open to illegal access/usage.

- **Organizations focused on PaaS**

Usage of Cloud Services works on a paid-for basis, which means that the finance department will want to store a record of how each of the services is being used. The Cloud Service Providers (CSP) themselves provide this information in the form of reports and logs, but in the case of a contention, it is always wiser to have an independent audit trail. For example, Cloud Service Providers like Amazon Web Services provides timely reports of your usage with customized policies (Snapshot below).

Report Details - Shweta

Data refresh settings ⓘ

Opted in

S3 bucket

shwetabucket1

Report path prefix

/Shweta

Time granularity ⓘ

Hourly

Report versioning

Create new report version

Compression type

GZIP

File format

text/csv

Report content

Account Identifiers

Invoice and Bill Information

Usage Amount and Unit

Rates and Costs

Product Attributes

Pricing Attributes

Cost Allocation Tags

Resource IDs

Figure 5 Snapshot of AWS Audit Reporting

Audit trails usually provide valuable and accurate information about an organization's employees' interactions with the cloud services. In order to create an independent audit trail of its own cloud service consumption, the end-user organization could consider a Cloud Service Broker (CSB) solution. Once equipped with his/her own records of cloud service activity, the CSO can confidently cater to any concerns over billing or to verify employee activity. A CSP should provide

the reporting tools to give organizations the power to actively monitor how services are being used. There are a plethora of reasons as to why an organization may want a record of Cloud activity, the main one being – To understand how the ‘cloud platform’ is being used for the organization’s business requirements, budgeting, and financial analysis.

- **Organizations focussed on IaaS**

The most important security consideration when the business is focussed on IaaS is protecting yourself from problematic cloud usage and redundant cloud providers. This is where Governance in Cloud computing becomes important. For instance, the organization may want to ensure that a user working in sales can only access specific sales and marketing leads and does not have access to other restricted departments/areas. Another example is that an organization, big or small, may wish to control how many virtual machines can be spun up by their employees, and, indeed, are the same machines spun down later when they are no longer needed. Instances wherein an employee is setting up their own accounts for using a particular cloud service needs to be monitored, detected, and brought under an appropriate governance umbrella. The image below from an organization account created in the Amazon Web Services gives an idea about how the access can be limited. The Root admin has the rights to decide which user is allowed to access the services:

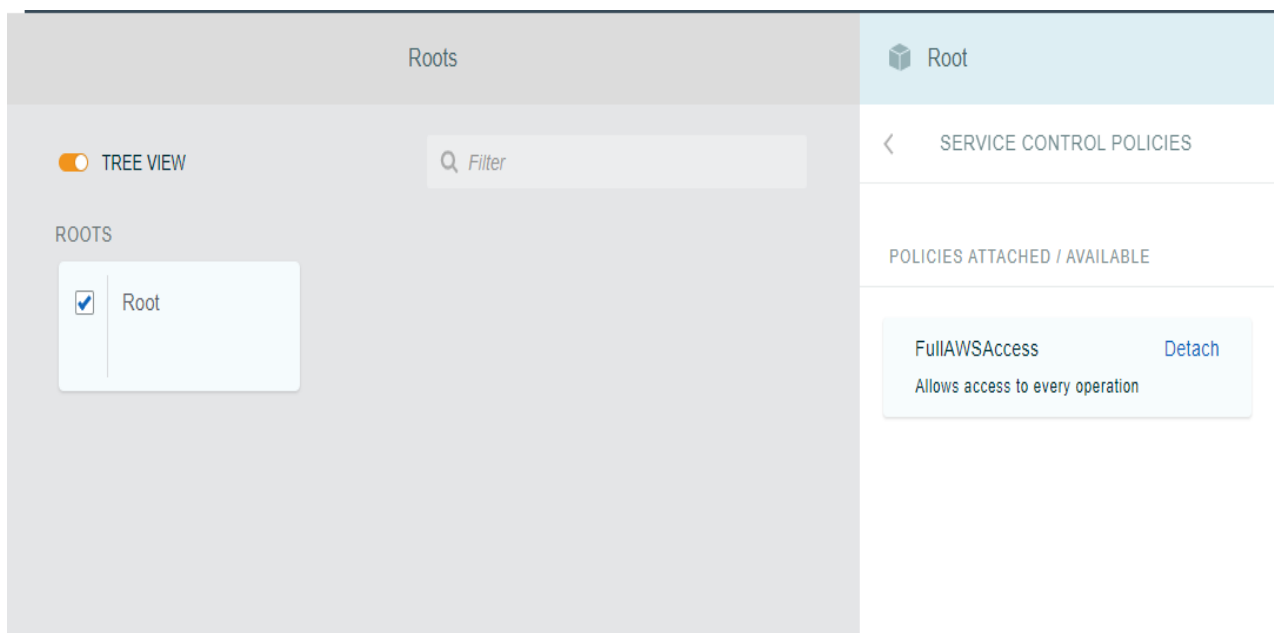


Figure 6 Snapshot of resource access (Resource Management)

While the Cloud Service providers (CSPs) provide different degrees of cloud service monitoring, an organization must consider implementing its own Cloud service monitoring and governance framework. The requirement for this independent control is of great benefit when an organization is using multiple SaaS providers, i.e., HR services, ERP, and CRM systems. However, in such a scenario, it is important to be aware that different Cloud Providers have unique methods of accessing and drilling information. They also have various security models on top of that to make sure that the data is secure.

Some use REST, SOAP, and so on. For security, some use certificates, and some use API keys. Some users simply use basic HTTP authentication. The main problem that needs solving is that these cloud service providers (CSPs) all present themselves very differently and uniquely. Hence, in order to use multiple Cloud Providers, organizations have to overcome the fact they are all different at a technical level.

Again, that brings us to the solution provided by a Cloud Broker, which brokers the several connections and essentially softens over the distinctions existing between the connections. This means organizations can now make use of several services together. Only in situations wherein there is something relatively commoditized like storage as a service; the services can be deployed interchangeably. This provides a solution for the issue solves of what to do if a cloud provider goes down or becomes unreliable. In fact, according to Mark O'Neil, CTO of Vordel, organizations should never have to get into the technical weeds of being able to comprehend, understand, or mitigate between different interfaces. They should be able to move up a level according to their requirements when they are actually using the Cloud mainly for the benefits of saving money.

- **For Businesses focussed on SaaS, PaaS, and IaaS**

Several Cloud services are accessed using simple REST Web Services interfaces. These are commonly called "APIs (Application Programming Interfaces)." They are analogous to the more heavyweight C++ or Java APIs used by programmers, but they are much lighter to incorporate from a Web page or from a mobile phone, hence their increasing universality. Another characteristic of using API keys is that they do not have a backend server. This means they do not identify the user or application making the API request, so there is absolutely no way to restrict access to specific users/accounts. How do they work? In order to identify authenticated users, the

user is pre-authenticated and issued an API key. The API requests include the API key, usually in a request header. They enable organizations to access the Cloud Provider like the AWS. For example, if an organization is using a SaaS service, it will often be provided with API Keys. Nevertheless, the protection of these keys is very important. Just like passwords, API keys must be regularly be rotated to prevent a compromised key from being abused.

Consider a good example of Google Apps. If an organization requires to enable single sign-on to their Google Apps (just so that their users can access their emails without actually having to log in a second time), then this access is given via API Keys. But if these keys were to be stolen, then an attacker would have access to the email of every person in that organization, and that could prove catastrophic for the organization.

Some systems have a way to issue API keys on demand. These keys may only be valid for hours or minutes, thereby limiting the exposure of a compromised key. Another method to protect API Keys is by encrypting them while storing them within a Hardware Security Module (HSM), or when they are stored on the file system.

The screenshot below shows us the options Amazon Web Services gives us to manage the API keys effectively. You can opt for self-management of the keys or entrust the job to the cloud provider.

Manage Encryption [X]

☒ **DEFAULT**

The key is owned by Amazon DynamoDB. You are not charged any fee for using these CMKs.

☐ **KMS - Customer managed CMK**

The key is stored in your account that you create, own, and manage. AWS Key Management Service (KMS) charges apply. [Learn more](#)

☐ **KMS - AWS managed CMK**

The key is stored in your account and is managed by AWS Key Management Service (KMS). AWS KMS charges apply.

[Cancel] [Save]

Figure 7 Snapshot of API Key management in AWS

CogniCrypt

One such method to secure API keys efficiently has been discussed in a research paper- CogniCrypt[134]. This proposed concept is typically described for programmers/developers, but I suggest the idea can be extended to securing API keys with the cloud providers as well. To begin with, the research paper aims to fulfill the following walloping gaps:

- a. Generate secure implementations for common programming tasks that involve cryptography (e.g., Data encryption). [134]
- b. Analyzes developer code and generates alerts for misuses of cryptographic APIs.[134]

The method:

The method for CogniCrypt has been devised on a series of surveys conducted among developers who found the usage of cryptographic APIs really difficult due to several reasons. The main reason was the usage of only a low-level cryptographic algorithm rather than more functionally oriented APIs. In the following flowchart, I have described the method CogniCrypt follows but customized it more for the usage in cloud services rather than programming. This step by step description (Figure 8), followed by the screenshots provided in the research paper, provides us a way to encrypt the APIs more efficiently and thereby increasing the security when using them.

Data related tasks supported by the CogniCrypt are:

- Symmetric Encryption
- Password Storage
- Secure Communication
- Secure Long-term storage.
- Secure Multi-party Computation

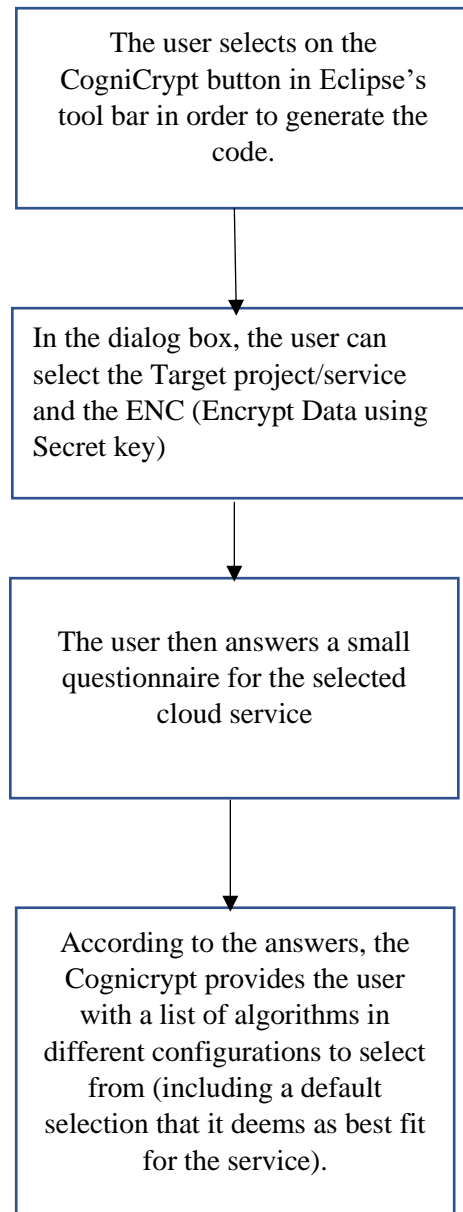


Figure 8 Flowchart- CogniCrypt method for Cloud Services

Attached below are two screenshots that portray the step 2 and step 4 of this process. The list of algorithms can be provided with a description of its usage so as to enable employees with a minimum knowledge in cryptography to understand what is best for the service they're trying to employ. This is, of course, in addition to a default algorithm selection by the CogniCrypt itself, describing why it is deemed the best fit for the application/service.

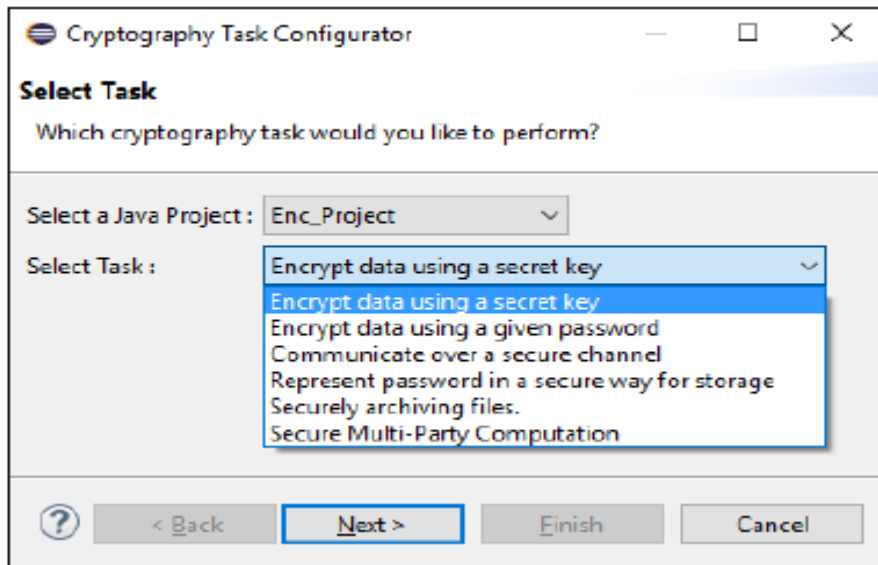


Figure 9 Selecting a Service

In the figure above, instead of selecting a development project, we could use a similar module to select a cloud service.

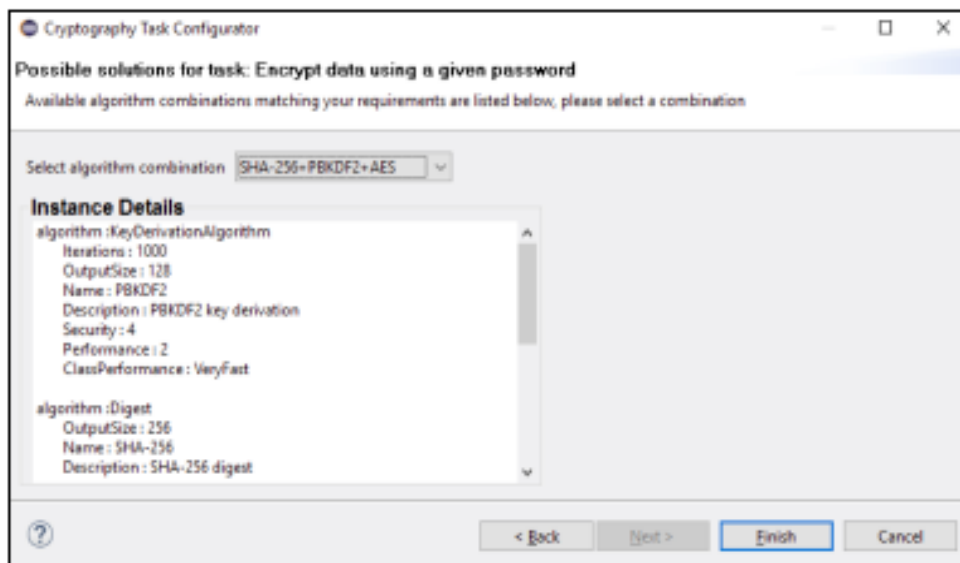


Figure 10 Selecting a cryptographic configurator

One other important feature of this method would be to report misuses as Eclipse error markers that can be rectified. The next question would be- How do we manage these cryptographic algorithms efficiently? The answer lies in deploying a Cloud API key management system. The provider employs a cloud-hosted

key management service that lets you manage encryption for your cloud services the same way you do on-premises. You can generate, use, rotate, and destroy cryptographic keys. In Google Cloud, a Cloud Key Management Service (Cloud KMS is) integrated with Cloud Identity and Access Management and Cloud Audit Logs so that the organizations can manage permissions on individual keys and monitor how they are used.

Understanding Cloud Computing

The National Institute of Standards and Technology in the United States defines cloud computing as “a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources that can be rapidly supplied and released with minimal management effort or service-provider interaction.”

Common characteristics of cloud computing include (Source: Deloitte [2])

1. Measured service

Cloud systems automatically control, monitor, and optimize resources, and they can also report usage, which provides transparency for both the Cloud Service Provider (CSP) and the consumer (Organization/business). Turn-key computing solutions shorten the ideation-to-implementation cycle. There are services provided by the AWS for monitoring, observing, and optimizing the hosted applications- Amazon CloudWatch Services. Bonus: These Cloud solutions operate on a subscription basis (either annually or quarterly), so there is no stress of long-term capital investments. Below is the screenshot of Amazon CloudWatch Services monitoring my usage and resources:

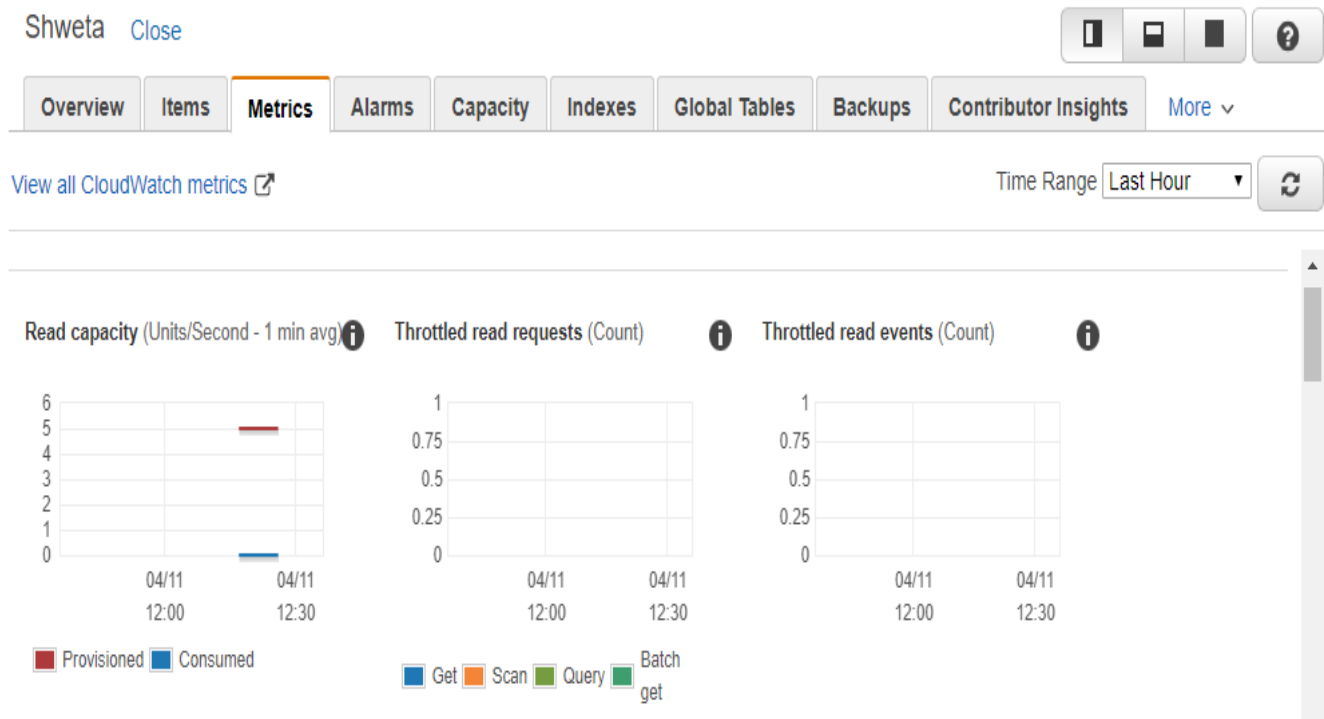


Figure 11 Snapshot of CloudWatch Metrics in AWS

There are several metrics that the CloudWatch covers- Latency, Time to live(TTL), system errors, write capacity, throttled write requests and events etc.

2. Rapid elasticity

Based on demand, resources need to be supplied and scaled rapidly. By being able to provide the extra technology resources whenever needed, organizations can accommodate sudden spikes in demand. Typically, there are two types of scaling that can be done:

- **Horizontal Scaling-** Adding or removing nodes, servers, or instances to or from a pool like a cluster. This is the most popular kind of scaling because it is more dynamic. For example, adding a load balancer for distributing requests to a web server.
- **Vertical Scaling-** Adding or removing resources to an existing node, server, or instance in order to increase the capacity of the node, server, or instance. This is the less used type of scaling since it is less dynamic owing to frequent reboots of systems, adding physical components to the servers, etc.

Today, we have three types of scaling- Manual, semi-automated, and fully automated(elastic). AWS provides services such as Amazon S3, Amazon Aurora, and Amazon SQS that provide elasticity as a part of their service for certain services/aspects of the services that are not elastic by design. Elasticity can be specifically implemented using AWS Auto Scaling or Application Auto Scaling. This is a fully automated type of scaling. It involves absolutely no manual labor to increase or decrease capacity. Below is the screenshot of the AWS Scaling Strategies available for my DynamoDB application service. According to my requirements, I can select the strategy:

Scaling strategy
The strategy defines the scaling metric and target value used to scale your resources.

<input type="radio"/> Optimize for availability Keep the read and write capacity utilization of your DynamoDB tables and indexes at 40% to provide high availability and ensure capacity to absorb spikes in demand.	<input type="radio"/> Balance availability and cost Keep the read and write capacity utilization of your DynamoDB tables and indexes at 50% to provide optimal availability and reduce costs.	<input checked="" type="radio"/> Optimize for cost Keep the read and write capacity utilization of your DynamoDB tables and indexes at 70% to ensure lower costs.	<input type="radio"/> Custom Choose your own scaling metric, target value, and other settings.
--	---	---	--

☒ **Enable dynamic scaling**
Support your scaling strategy by creating target tracking scaling policies to monitor your scaling metric and increase or decrease capacity as you need it. [Info](#)

Figure 12 Snapshot of selection of Scaling Strategy in AWS

Once you’ve selected the scaling strategy and the metrics required for your application, a scaling plan is created that is customized to your needs. In addition to creating a plan and monitoring it (see **figure 13**), you can also add this scaling plan to your CloudWatch dashboard so as to monitor all your resources in a single place.

table/Shweta

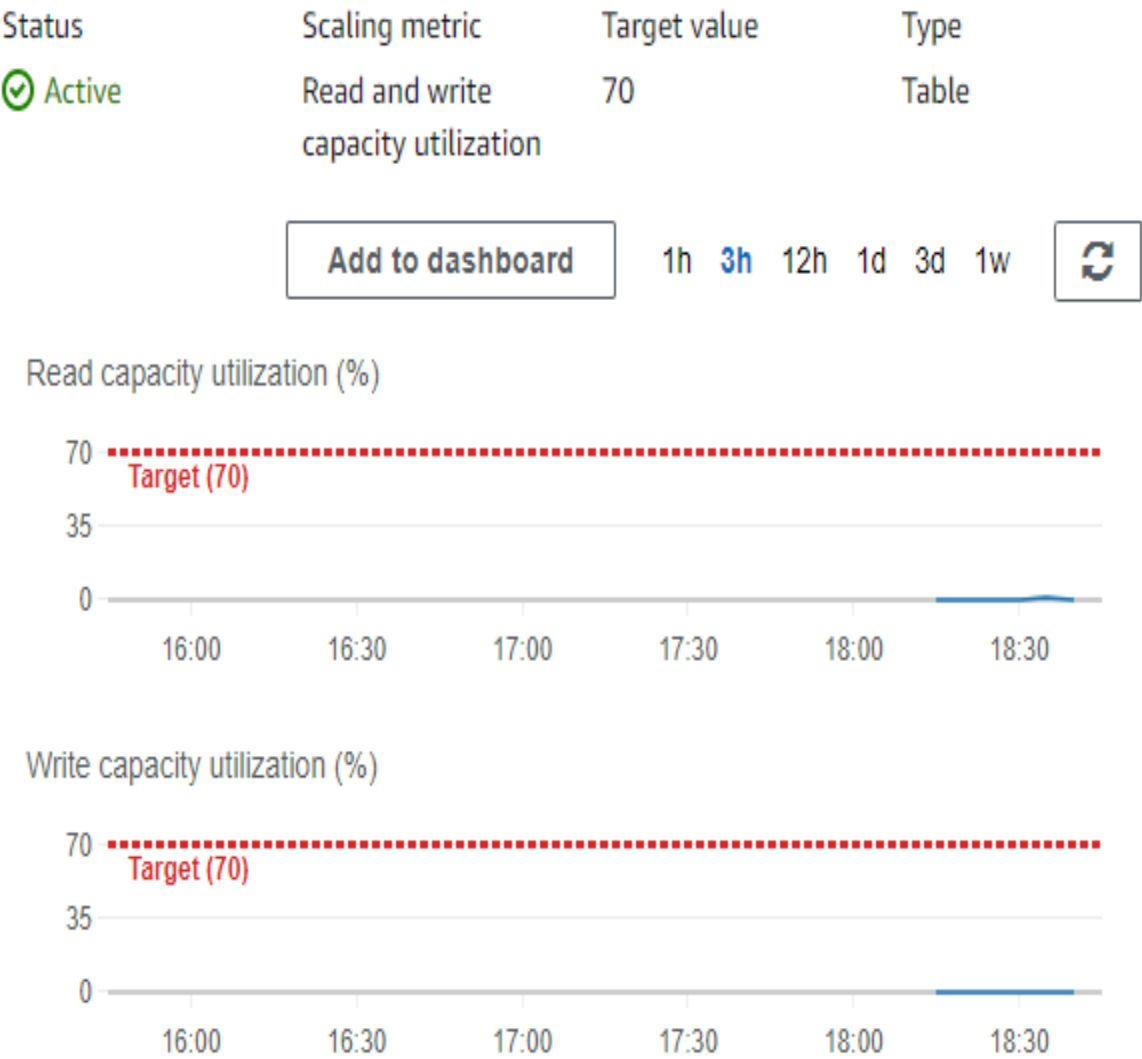


Figure 13 Snapshot of CloudWatch Dashboard in AWS

3. On-demand self-service

Users can go on to provide their own computing capabilities as required with each service provider, with absolutely no need for human interaction. The ongoing and fast-evolving nature of cloud expenditures encourages organizations to focus all their time, energy, and effort, resources on using IT services rather than conducting extensive planning exercises to secure funding and/or implement IT infrastructure [2].

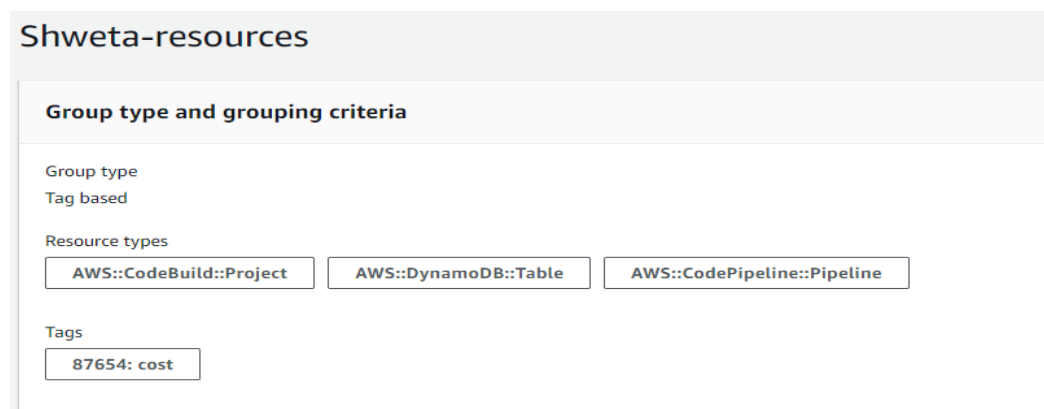
4. Broad network access via all platforms

A complete access is available over the network and through various client platforms such as mobile phones, tablets, laptops, and workstations).

5. Resource-pooling in Cloud

The Cloud Service provider's computing resources are pooled to serve multiple consumers, with resources reassigned and customized according to demand. Cloud Service Providers (CSPs) introduce new features and functions in the market on a very regular basis; in fact, some have known to have launched over a thousand features annually. Additionally, they give organizations the leverage and freedom to redirect IT roles toward developing capabilities in such areas as machine learning, the Internet of things (IoT), and artificial intelligence (AI) rather than focus on operations [2]. Amazon Web Services (AWS) provides what is called AWS Resource Groups to organize and automate all the AWS resources at a time.

In this image below, I have created a resource group in the Amazon Web Services, selected three resources, and an API key (tag) with the value-cost. Several such resource groups can be created, and access can be restricted/permitted based on the tags used.



The screenshot displays the AWS Resource Groups console for a resource group named "Shweta-resources". The "Group type and grouping criteria" section is active, showing a "Tag based" group type. Under "Resource types", three resources are selected: "AWS::CodeBuild::Project", "AWS::DynamoDB::Table", and "AWS::CodePipeline::Pipeline". Under "Tags", a tag with the key "87654:" and value "cost" is specified.

Group type and grouping criteria
Group type Tag based
Resource types AWS::CodeBuild::Project AWS::DynamoDB::Table AWS::CodePipeline::Pipeline
Tags 87654: cost

Figure 14 Snapshot of Resource Groups- AWS

Case Studies on security breaches:

Below are some top security breaches that have affected big companies in the past leading to loss of data from the cloud. (Source: Cyber Security hub [26])

Company/Industry type	Records exposed	Type of attack	What can we do better?
Capital One-BFSI	1066 Million	Cloud Vulnerability	Regular assessments of firewalls and restricted access to configuration settings
State Farm Insurance-BFSI	N/A	Credential Stuffing	Augment security awareness training as to why 'unique credentials' are important, Implement multiple forms of authentication.
Dunkin Donuts- Restaurants and Hospitality.	N/A	Credential Stuffing	Implement Good password hygiene and two-factor authentications.

Biostar 2- software and Technology	1 million	Cloud Vulnerability	Third-party risk assessments for SaaS and PaaS providers make alternate authentication methods available.
------------------------------------	-----------	---------------------	---

Understanding Cloud Security and Data Privacy

These days, businesses have completely shifted from assessing whether they should use cloud computing to plan how they are going to incorporate it as an important part of their IT infrastructure. Regardless of all the advantages of storing data in the cloud, it is highly critical to understand, realize, and mitigate risks related to moving corporate data out of a carefully built and secure on-premise environment. Losing complete operational control of such data may make it vulnerable to external security breaches and privacy threats, if adequate safeguards and threat controls are not put into place. Furthermore, public cloud architectures are dynamic in nature, which can make security and privacy measures, both cumbersome and expensive.

Main Causes for Security Breaches:

Limited organizational control of cloud consumption

It is possible for stakeholders to procure cloud services with little to absolutely no oversight from their IT and procurement teams, which has led to several organizations assuming a greater level of risk than with traditional IT procurements. Cloud systems are typically characterized by the providers having control over the physical premises, hardware, software, and networks and not particularly dedicated to a single business/ customer. Common operational, technical standards, and security policies are used for all customers/businesses. SaaS applications are all pre-configured and are difficult to customize to the particular needs of a business. Due to this lack of control and inability to directly control the used services, it is not a herculean task for an attacker to breach into a company's data [2].

Limited negotiation with cloud service providers

Cloud Service Providers (CSP) are typically not very open to negotiating and explaining their standard terms and conditions entirely. Various kinds of clients use public cloud environments for different services, and providers don't still offer services customized for unique or specialized requirements of the users. This thereby contributes further to the inability to control security regulations [2].

Rapidly changing landscape

With a relatively new market offering, cloud solutions are still maturing in terms of industry standards

and operating models. Regulators are also continuously evolving, and many providers don't still provide clear guidance regarding cloud computing and its services. As various industry groups, alliances, and government agencies continue to develop their own unique standards, organizations must make sure that CSPs are keeping pace with compliance and regulatory changes. For instance, the Internet of Things (IoT), Blockchain technology require more customization and focus on the back end of cloud computing [2].

Inability to Monitor data in transit

This is, by far, one of the major drawbacks in cloud computing. This provides a platform for the attackers to compromise data while in transit. Attacks like Man-in-the-middle (MITM), MITM Secure Sockets Layer, DNS Spoofing, and several overflow-based attacks are all common attacks affecting the cloud services today.

Inability to prevent insider theft

For security regarding sensitive business data and sanctioned enterprise cloud applications, Insider Threat, a threat that comes from an individual/employee with a legally authorized access to an organization's IT systems becomes a natural priority. Whether malicious or accidental, security incidents involving insiders can put large amounts of highly sensitive data at risk leading to a great financial burden for businesses to recover from. Even when IT security teams may consider data "secure" within sanctioned cloud applications, it may not really be the case. According to MacAfee, Grand Theft Data: Data Exfiltration Study, Insiders are responsible for 43% of all data breaches, but there is a general consensus across the security industry that breaches attributed to insiders tend to be more harmful to the organization. The majority of the breaches from insider threats involve malicious intent, while only 28% are really accidental. This area of risk most frequently involves current and former employees/users, but external contractors, auditors, and consultants can also put data at risk. Given the high economic losses and difficulty of detecting incidents, addressing Insider Threat is a key element of any organization's cloud security strategy.

Studying the Cloud Security with an example- Amazon Web Services

Below are some of the tactics and techniques representing the **MITRE ATT&CK Matrix** for Enterprise

covering cloud-based techniques, particularly in Amazon Web Services:

Initial Access	Persistence	Privilege Escalation	Defense Evasion	Credential Access	Discovery	Collection	Exfiltration	Impact
Exploit Public-Facing Application	Account Manipulation	Valid Accounts	Redundant Access	Account Manipulation	Cloud Service Dashboard	Data from Cloud Storage Object	Transfer Data to Cloud Account	Resource Hijacking
Trusted Relationship	Create Account		Revert Cloud Instance	Cloud Instance Metadata API	Cloud Service Discovery	Data from Information Repositories		
Valid Accounts	Implant Container Image		Unused/Unsupported Cloud Regions	Credentials in Files	Network Service Scanning	Data from Local System		
	Redundant Access		Valid Accounts		Network Share Discovery	Data Staged		
	Valid Accounts				Remote System Discovery			
					System Information Discovery			
					System Network Connections Discovery			

Figure 15 List of Attacks on Cloud

We will now discuss some of these attacks in detail and understand how it affects the cloud security.

Valid Accounts- Initial Access

The credentials of a specific user or a service/business account can be stolen by an attacker using Credential Access techniques or capture credentials through social engineering for means of gaining Initial Access. What are some techniques used for gaining Credential Access, and how can they be mitigated in order to nip it at the bud? The following table explains three main techniques:

Name of the Attack	Description	Mitigation
Link-Local Multicast Name Resolution (LLMNR/NetBIOS-Name Service(NBT-NS) Poisoning	LLMNR/NBT-NS provides information for internal systems that may not be easily accessible. Poisoning this service can easily allow the	Improving password policy, Ensuring SMB signing is enabled, putting security policies in place to combat any issues in the LLMNR/NBT-NS

	attacker to gain access to the credentials.	traffic.
Kerberoasting	Kerberos is a windows network authentication protocol that utilizes tickets to allow strong authentication between devices. Within a windows environment, there are Service Principal Names (SPNs), which are utilized to identify each instance of a Windows Service. Attackers can request service tickets for any SPNs from a domain controller. The service ticket that is generated contains the hashed password for the service accounts associated with the SPN. The ticket can then be subjected to offline password recovery attacks to gain the credentials to that service account, after which the attacker can easily escalate his/her privileges.	Enabling strong encryption algorithms specifically for Kerberos, ensuring service accounts have only a minimum level of access, implementing strong password conditions.

Credential dumping	Credential Dumping is the means of gaining access to the account login and password either in cleartext or hashed format from a system that has been compromised. For Windows, attackers usually try to manipulate the registry that interacts with the NTLM credentials within the memory. For Linux, the attacker tries to scrape live memory out of other programs running in the system.	Up to date operating system and software patches, complex passwords for the root access, Protection of the Security Account Manager (SAM) database.
--------------------	--	---

Accounts that an attacker may be classified into fall into three main categories: default, local, and domain accounts. **Default accounts** are defined as those accounts which are built-into an OS such as a default factory set accounts on other types of software, systems, or devices or a simple Guest or Administrator account on Windows systems. **Local accounts** are those configured by an organization for use by users, remote support, services, or administration on a single system or service. **Domain accounts** are those accounts that are managed by Active Directory Domain Services wherein the permissions and accesses are configured across services and systems that are part of that domain. Domain accounts can cover administrators, users, and services[38].

Credentials that are compromised can be used to bypass access controls that are placed on various resources within the systems in the network and may even be used for persistent/redundant access to externally available services and remote systems, such as Outlook Web Access, VPNs and remote desktop. Compromised credentials may also allow an attacker increased privilege to specific systems. Attackers may also utilize publicly disclosed private keys, or stolen private keys, to legitimately connect to remote environments via Remote services.

The overlap of all the account access, credentials, and permissions across a network of systems is of a major concern because the attacker may be able to gain across accounts and systems to reach a high level of access (i.e., domain or enterprise administrator) and can then easily bypass access controls set within the enterprise leading to a major data breach and huge financial losses.

Mitigation:

Apart from using multi-factor authentication and strong password policies, we have something known as **Filter Network traffic**, which is one of the effective ways to combat this type of attack in the cloud. It involves whitelisting IP addresses on the cloud along with user account management to ensure that the data access is restricted only to legitimate users. This method makes sure that the ‘legitimate’ users are from the expected IP range only, thus, serving as a second round of a check post.

One example of implementing the Filter Network traffic on Amazon Web Services is given below:

This shows how to whitelist IP addresses to access the Amazon API Gateway. The procedure is very simple:

- a. You choose an API (HTTP, WebSocket, REST)
- b. You can then choose “Create Method.” You can choose from POST, DELETE, PATCH, etc.
- c. You can then need to create a resource policy that describes how you want your filter to work-permit/restrict IP addresses.

```

1 {
2   "Version": "2012-10-17",
3   "Statement": [
4     {
5       "Effect": "Allow",
6       "Principal": "*",
7       "Action": "execute-api:Invoke",
8       "Resource": "arn:aws:execute-api:ca-central-1:648810493763:bv7596bela/*//*"
9     },
10    {
11      "Effect": "Deny",
12      "Principal": "*",
13      "Action": "execute-api:Invoke",
14      "Resource": "arn:aws:execute-api:ca-central-1:648810493763:bv7596bela/*//*",
15      "Condition": {
16        "NotIpAddress": {
17          "aws:SourceIp": "192.168.56.12"
18        }
19      }
20    }
21  ]
22 }

```

Figure 16 Snapshot of a Resource Policy code-AWS

This is a policy written to allow access to the mentioned IP address. You can also give a range of addresses in the Classless Interdomain Routing format.

- d. Save the policy and deploy the API.

This deployed API then makes sure to filter access to the services based on the policy.

Redundant Access- Persistence

If one kind of illegal access tool has been detected, blocked and removed by an organization in response to a security breach attempt but the organization did not completely understand the attacker's tools and access methods, then the attacker can retain access to the network services. This is known as Redundant Access. External Remote Services such as external VPN can be used to maintain access to some valid accounts despite interruptions. Another method used by the attackers to maintain access is using Web Shell.

“A Web Shell is a web script that is placed on an openly accessible web server as a gateway to the network.” A Web shell provides a set of functions that can be executed or a command-line interface (CLI)

on the system that ideally hosts the Web server. In addition to a server-side script, a Web shell may also have an additional client interface program that is utilized to talk to the Web server directly. One real-life example of this would be the China Chopper.

The China Chopper is said to be just 4kb in size with a very easily modifiable payload. It has two main components that are run by different adversaries- the China Chopper Client (run directly by the attacker) and the China Chopper Server (installed on the victim's web server- more like a tracking chip, installed in plaintext). This server will be controlled by the attacker but indirectly through the victim's web server. It can issue commands and manage everything on the victim's server. In fact, it's hash is publicly available:

Web shell client (caidao.exe) == 5001ef50c7e869253a7c152a638eab8a (MD5 hash)

One of the basic ways to prevent this intrusion would be to regulate timely patch and software updates.

Mitigation

Redundant Access can be prevented by a method known as **Network Intrusion Prevention**. This method uses **network signatures** to identify traffic for a specific type of malware. Each family of malware uses unique signatures within their protocols. The signatures are generally of two types:

- **Atomic Signatures**- it is the simplest type of signature that is triggered on a single event. They do not ideally need the intrusion system to maintain state, and the inspection does not require the knowledge of past or future activities. For example, the LAND attack- a type of DDoS attack, by inspecting a single packet, the network/host-based intrusion system can identify the attack because no state information is needed.
- **Stateful Signatures**- They, on the other hand, are triggered on a sequence of events that ideally require the Intrusion Prevention System (IPS) to maintain state.

Revert Cloud Instance- Defense Evasion

An attacker may regress the changes made to a cloud instance after they have successfully performed malicious/illegal activities in an attempt to bypass the detection and erase any evidence of their presence. In highly virtualized environments like the cloud-based infrastructure, this may be easily done using

restoration from a Virtual Machine or data storage screenshots through the cloud services management dashboard itself. Another use of this technique is to take advantage of temporary storage, which is attached to the compute instance. Most cloud providers provide various types of storage, including persistent, local, and/or ephemeral, with the latter types often reset upon stop/restart of the Virtual Machine.

Mitigation

The best way to prevent this attack would be to establish centralized logging of instance activity and subscribe to regular reports from the provider, which can be used to monitor and review system events even after reverting to a screenshot, rolling back changes, or changing persistence/type of storage. Monitor specifically for events related to screenshots, rollbacks, and Virtual Machine configuration changes, that are usually occurring out of the ordinary. To reduce the false positives, valid change management and incident management procedures could benefit by introducing a known identifier (tag) that is logged with the change (e.g., tag or header) if supported by the cloud provider, to help differentiate between valid, expected actions from malicious ones.

Below is the example of one such monitoring and reviewing system with Amazon web Services- Amazon CloudTrail. You can view the consolidated list of events encountered in the selected time period, the source address, and the users. You can view it in the form of tables or trails. This feature also gives us insights on the usage. It also employs the use of tag and access keys to ensure effective change management.

Filter:	Read only ▼	false	Time range:	Select time range
	Event time	User name	Event name	Resource type
▼	2020-04-11, 04:58:56 PM	AWSOrganizations	CreateServiceLinkedRole	
AWS access key ASIAZOE605BZC2ZEGO2				
AWS region us-east-1				
Error code InvalidInputException				
Event ID 5af273d4-164d-4f7e-8d10-c93e8f5ecd8e				
Event name CreateServiceLinkedRole				

Figure 17 Snapshot of amazon CloudTrail events- AWS

The screenshot below is the CSV file you can download from the cloudTrail in order to save your tracked event information offline.

Event ID	Event time	User name	Event name	Resource type	AWS region	Source IP address
23ffa1d3-	2020-04-11	AWSOrganizations	CreateServiceLinkedRole		us-east-1	organizations.amazonaws.com
5af273d4-	2020-04-11	AWSOrganizations	CreateServiceLinkedRole		us-east-1	organizations.amazonaws.com
029a0505-	2020-04-11	AWSOrganizations	CreateServiceLinkedRole		us-east-1	organizations.amazonaws.com
4d6523e4-	2020-04-11	root	PutEventSelectors	CloudTrail Trail	ca-central-1	68.150.220.130
56a9848e-	2020-04-11	root	StartLogging	CloudTrail Trail	ca-central-1	68.150.220.130
5faf6eea-	2020-04-11	AWSOrganizations	CreateServiceLinkedRole		us-east-1	organizations.amazonaws.com
7a3bdf5-	2020-04-11	root	CreateTrail	CloudTrail Trail and 1 more	ca-central-1	68.150.220.130
8a7c6257-	2020-04-11	root	PutInsightSelectors		ca-central-1	68.150.220.130
bf312cc8-	2020-04-11	root	CreateServiceLinkedRole		us-east-1	cloudtrail.amazonaws.com
dd2c9a4f-	2020-04-11	root	CreateDeployment		ca-central-1	68.150.220.130
97223585-	2020-04-11	root	UpdateRestApi		ca-central-1	68.150.220.130
90687be2-	2020-04-11	root	CreateDeployment		ca-central-1	68.150.220.130
d6313a7b-	2020-04-11	root	UpdateRestApi		ca-central-1	68.150.220.130
3990294c-	2020-04-11	root	CreateDeployment		ca-central-1	68.150.220.130
72d959b8-	2020-04-11	root	UpdateRestApi		ca-central-1	68.150.220.130
eeaa5b36-	2020-04-11	root	UpdateRestApi		ca-central-1	68.150.220.130
b2d2b26f-	2020-04-11	root	PutIntegrationResponse		ca-central-1	68.150.220.130
de220ee2-	2020-04-11	root	PutIntegration		ca-central-1	68.150.220.130
6b823e9f-	2020-04-11	root	PutMethod		ca-central-1	68.150.220.130
abcb606d-	2020-04-11	root	PutMethodResponse		ca-central-1	68.150.220.130
2ea15835-	2020-04-11	root	ImportRestApi		ca-central-1	68.150.220.130
7bf61bfa-	2020-04-11	root	ListContributorInsights		ca-central-1	68.150.220.130

Figure 18 CSV file of the CloudTrail logs from AWS

Basic Preventive Measures to Avoid Security Breaches in Cloud

Well-defined organizational controls related to cloud purchases and use

With an enterprise risk-management perspective in mind, companies should develop and implement cloud-specific procurement guidelines with required and preferable terms and conditions of the cloud provider as well as create the important necessary mechanisms to enable a well-engineered transformation to the cloud [2].

Cloud vendor landscape evaluations

Organizations must authorize the fundamental guidelines around regulatory and compliance requirements for the services offered by the cloud, which will thereby provide a baseline for several vendor evaluations. It's extremely important to understand what to look for in terms and conditions and determine and agree upon what constitutes an acceptable level of risk based on business priorities and the organization's resources. When evaluating cloud vendors, the organization must look into baseline compliance requirements such as **data protection and incident response, user identity and access management, and data residency requirements**, among others [2][1].

Update your Bring Your Own Device (BYOD) Policy

Nowadays, It has become a norm and rather 'cool' to bring your own devices to work. In many cases, businesses(like small startups, small scale businesses) can benefit from this as they don't have the responsibility to supply as many tech devices for employees. However, by permitting these 'outside' devices into the workplace, you are also unknowingly opening the door for data security breaches and loopholes through several new end-point devices. Updating the BYOD (bring your own device) policy will go a long way in ensuring that that all devices brought into the organization premises are following the same data security procedures as your in-house tech. It is clearly important to be aware of any work-related information that might get shared on personal email accounts, as this can lead to leakage of sensitive information intentionally or unintentionally.

Social Engineering Training

Social Engineering is defined as a practice of manipulating people in order to get them to give out information or take action. Even the most tech-savvy employees in the organization can easily fall for social engineering attacks if done smartly.

Suppose A hacker has a set of email addresses then he can possibly send some luring messages like “Hello, Mr. XYZ, We are glad to inform you that you have won 1200\$. Only one step far for claiming this reward. Register your email id with us and claim your reward”. This is a type of phishing method to retrieve details. This email will be made to look like it has been sent from a well-known organization or even from your own company. But in reality, it leads the user to a malicious link. Clicking on that link may land you in a page that may look like a legitimate Facebook page, and people might actually end up submitting the required details to that fake page. This is known as “Spoofing Attack.” If a person’s computer is not secured with a firewall, it becomes extremely easy for the attacker. This is exactly how social engineering works.

As users have gotten much smarter and more cautious about online scams, hackers have had to up their game. Sharing the details of professional lives should be significantly addressed in the workplace as its the primary way of collecting data by the hackers.

Have a well thought through and executable emergency plan

You may think that a data breach can never take place in your organization owing to the security regulations you currently have in place and that you may have the tightest data security services deployed, but the reality is if you do not have an emergency plan, you leave yourself vulnerable. It is extremely crucial to have a security plan in place. This is known as Cyber Risk Assessment (CRA) coined by the National Institute of Standards and Technology (NIST). The main idea behind incorporating CRA is to support proper risk responses. Informed decisions need to be taken by the company in order to prevent or combat such situations. It is usually based on the following points:

- Researching about the relevant and plausible threats to the organization.
- The vulnerabilities of the organization- both internal and external.
- The impact analysis of the vulnerabilities if ever exploited.
- The probability of the vulnerabilities being exploited.

Another prime reason to incorporate the CRA is that it is mandatory if you want to get cyber insurance without which companies can be put out of business too. Sometimes, it is also compulsory under the organization's compliance regulations. Usually, every company documents all the emergency response preparedness rules to be followed and circulates the guide to its employees. One such breach incident response guide by DLA PIPER lists some very basic preparedness guidelines that one can follow to understand [47]:

- Assemble an Incident/Breach Response Team (consisting of a first responder).
- Contact the council both inside and outside to establish an entirely privileged communication and reporting channel.
- Co-ordinate with the legal counsel to bring in forensic examiners and cybersecurity experts(Forensic Experts)
- Stop any kind of additional data loss.
- Secure Evidence (tokens, key cards, building credentials, logs and surveillance tapes)
- Preserve Computer logs (log files, including firewall, VPN, mail, network, client: web, server, and intrusion detection system logs).
- Document the breach.
- Contact the law enforcement.
- Define legal obligations.
- Conduct interviews of personnel involved.
- Reissue or Force Security Access Changes.
- Do Not Probe Computers and Affected Systems.
- Do Not Image or Copy Data, or Connect Storage Devices/Media, to Affected Systems.
- Do Not Run Antivirus Programs or Utilities.
- Do Not Reconnect Affected Systems.

Listed above is just a consolidated list of basic guidelines to be followed in case of an emergency breach by that particular industry [47]. Most organizations circle around the same ideas customized according to the size of the company.

To understand the current scenario and the future of cloud security, let us rummage into the literature review, wherein, we discuss some research papers slowly leading into the possible ways to increase the security in the protocol level:

Literature Review

1. Verifying Cloud Services- Present and the Future

Introduction

Since the underlying idea of the proposed method of this project deals with allocating more control over the sent packets to the sender, let us begin with a research paper that focuses primarily on identifying gaps in the existing cloud technology in terms of verification tools provided to the users. Very limited tools are currently available for clients to monitor and evaluate the provided cloud services. It is only equitable for the consumers who subscribe and pay for a service to expect it to be (among other features) available, secure, and reliable.

In this paper, they had given us an example of Amazon Elastic Cloud that faced an outage in 2011 when it crashed due to creating too many new backups of its storage volumes. Many large customers of Amazon, such as Reddit and Quora, were down for more than one day (Profit/loss calculation can be found in the discussion section).

Intuit experienced repeated similar outages in 2010 and 2011. No explanation had been provided to customers, who, for long, could not access their financial data.

Problems of **Authentication and authorization** have also given sleepless nights to companies: in 2011, DropBox admitted that a bug in their authentication mechanisms had disabled password authentication; hence, for four hours, the accounts of Dropbox's 25 million users could be accessed with literally any random password. Similar authorization issues have affected other cloud service providers, including Google[1].

What do you suggest is the best approach to tackle this problem? I would suggest **Identity and Access Management (IAM)**. This particular practice provides effective security, authentication, authorization, and provisioning of storage and verification. I came across this Flow diagram that depicts the taxonomy of cloud services security- You can also see the several authentication methods that are incorporated :

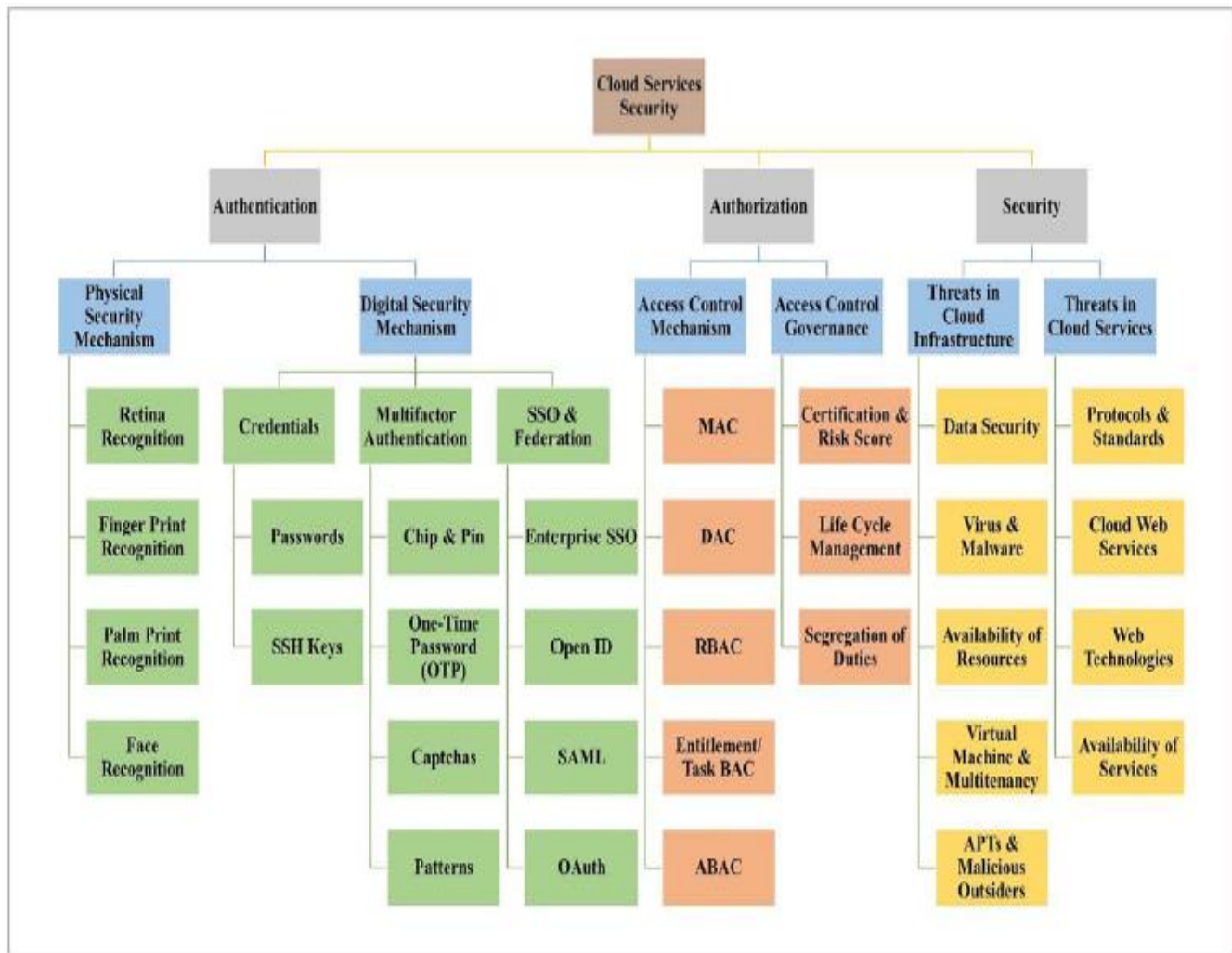


Figure 19 Taxonomy of Cloud Services Security [27]

Users could benefit from knowing to what extent a cloud provider delivers the promised service. More concretely, the contract between the cloud and its users should be verifiable (to some extent), and the sheer ability to detect failures, without relying entirely on the cloud provider's report, can be useful to the users. For example, it may be important to promptly find out that a service does not respect its functional specification; or that it generously shares personal data with the world, or that it is down, underperforms, or if its basic security controls seem to be failing. This information can be especially helpful for critical

applications such as medicine or banking and facilitate their process of adopting cloud technologies. In addition, verification tools to check these aspects can help consumers pick and choose a particular cloud provider.

However, this paper delves into the idea that cloud users may benefit from understanding in breadth, rather than only in-depth, whether they could verify service provisions in the cloud. They discuss recent research advances and propose directions for future research to help and bridge those gaps. They consider two main types of cloud customers- **service providers** deploying their software for execution in the cloud (with Platform-as-a-Service or Infrastructure-as-a-Service), and **cloud users**, who use a software or storage service executing in the cloud, be it provided by a third party software provider or by the cloud provider[1].

Methods implemented-

- Verification of strong service identities:

This is done by creating a possible path towards enabling service providers to analyze the deployed services and check for compliance with their original service implementation. The idea is to bind a strong service identity to the service instances on the cloud such that this unique association is preserved throughout the entire service lifecycle, from deployment to decommissioning. The authors focus on a promising implementation of this idea based on Trusted Computing and trusted servers. Cloud nodes run special software stacks -trusted software systems that can host the service instances in special environments, isolated from both the administrator and other tenants.

Cloud nodes are also equipped with commodity trusted computing hardware, which validates the integrity of the software stack upon boot and enables service providers to verify that the nodes are running a trusted software system. If this is the case, service identity is preserved.

They have implemented this by the following algorithm:

S = the service software implementation produced by the service provider

S_0 = an instance of the mentioned software service S that is hosted in the cloud.

A Service Identity is said to be strong if and only if the invariant $S = S_0$ holds constant for the entire lifecycle of S and in all the nodes where S is instantiated.

The lifecycle of a service spans the period between its deployment until its decommissioning. Throughout this lifecycle's length of time, the service might be migrated or replicated across various nodes on the cloud.

In Infrastructure-as-a-Service (IaaS), the service is generally deployed as a Virtual Machine Image (VMI) and instantiated in virtual machines (VMs).

In Platform-as-a-Service (PaaS), the service is shipped as an application package and instantiated into objects in application containers. A cloud platform utilizes and provides trusted containers to enforce a strong service identity.

The state of a service instance hosted by a trusted container, usually in isolation from other tenants and from the cloud administrator. This protection is enforced throughout the lifecycle of the service. While replicating or migrating service instances to other nodes, the trusted container transmits any relevant data and service code to the sink over an encrypted channel after verifying that the sink is also a trusted container. The cloud service provider also has the capability to verify that the target host offers trusted container protections before deploying the service.

A trusted software system is crafted in such a way that neither the tenants nor the administrator has any access to the service instances' state, which offers a specific hosting abstraction. To prevent man-in-the-middle attacks, the TPM (Trusted Platform Module) signs the registers' content with the private part of a cryptographic key pair that never leaves the TPM in plaintext. The remote party can then verify the signature and the content of the TPM registers using a public key certificate given by the cloud provider: If the trusted software system boots on the cloud node, its respective hash will show up in the TPM's registers.

- Verifying Functional properties on Cloud Services:

The authors propose a new approach allowing the users to verify service integrity in a scalable fashion without relying on either a centralized certification authority or access to the actual implementation code. Their approach is based on the decomposition of the verification process into three separate phases namely: test suite generation, test suite execution, and validation of the results, wherein each phase can be performed at a separate location in order to maximize exhaustiveness and the performance of the verification process. Their current proposal for incorporating the test suite generation is entirely based on the black-box testing techniques that generate test suites examining all the described functionalities.

- Verifying performance and Dependability properties:

The authors call for a new cloud model that integrates service levels and SLA (Service Level Agreement) effectively into the cloud. This proposed approach intends to guarantee and combine multiple cloud service level objectives in a flexible and consistent way. It allows us to engage in a better cloud QoS (Quality of Service) via a control-theoretic approach for controlling and modeling SLA-oriented cloud services. This paper also further discusses the ways to assess cloud service QoS guarantees and also help several system designers build and deploy SLA-oriented clouds that can be easily controlled.

- Verifying Security policies:

Cloud providers and consumers are often made to believe to consider encryption as the only security tool for sufficient protection while using the cloud services. Yet, encryption is not the solution for everything: resources and data are shared, schemes implemented to control the access to these resources at times fall short in the face of malicious insiders, applications are outsourced, or system misconfigurations take place. Moreover, when any of this happens, users are often expected to provide proof of the security violations they incurred. The authors, therefore, strongly believe that users may benefit from adding some tools that may allow them to

(1) Comprehend the security policy according to their needs.

(2) Have access to ways to gain evidence in case violations happen.

And (3) given an option to choose a better provider in case such violations are too frequent.

Discussion:

This research paper ideally answers the following questions and thus aims to fill in the gaps in the operation of cloud technology:

1. What may the delivery parameters of a cloud service be of interest to consumers?

From the discussion in this research paper, I would conclude the primary delivery parameters to be availability, reliability, security, and functional correctness and preferably in this particular order. Why, you ask? Let us take the example mentioned in the introduction section about big businesses such as Reddit or Quora going down for more than a day-). For the sake of discussion,

let us roughly calculate how much loss is incurred by a company when such a mishap takes place let's say, you have 4 hours of peak traffic per day, and the company generates about \$800,000 of total profits per week (say, 4 peak hours *7 days of the week= 28 hours). By dividing the total profit of \$800,000 by the number of peak hours/day, which says 4 hours in our case, you get approximately \$114,285 per day during peak hours only. When a large-scale business is down for over a day, you can imagine the amount of loss they have to incur. This example clearly goes to shows how critical **Availability** and **reliability** is for a service.

2. How can cloud providers guarantee a strong identity between the service implementation and the software running on the cloud nodes?

I came across this diagram used in this research paper that depicts the verification framework for services in a cloud.

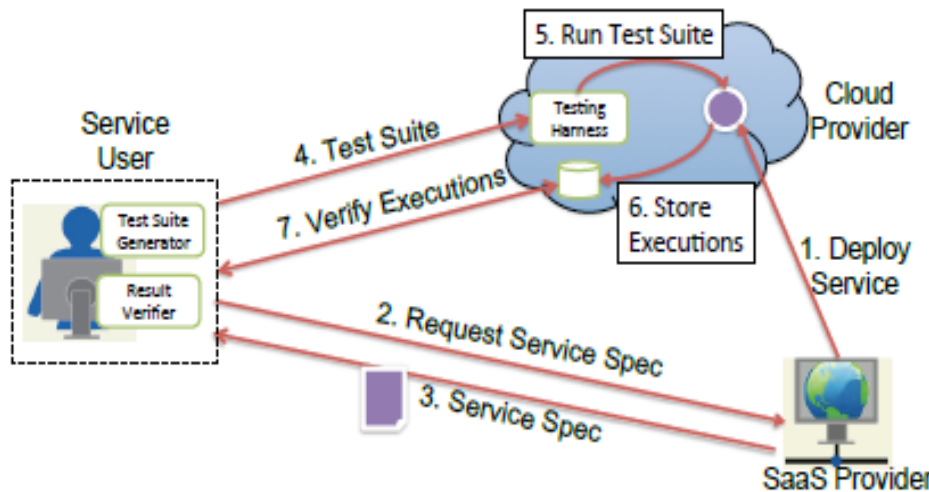


Figure 20 Proposed Verification framework for service Identities

Here, the service implementation is provided by the SaaS (Software as a Service). It advertises the service specifications to the user who, in turn, selects the one closest to his/her requirements. The selected specification is analyzed by the Test suite Generator to generate the best suite, which in turn is then given to the testing harness to deploy the service on the execution platform (Cloud). The results are stored in the cloud storage facilities. In this way, the users seem to get more control of what service they use and how it is implemented.

3. How can clients assure that their data is still stored somewhere in the cloud and not lost by a provider trying to cut storage costs?

A bigger business would mean more data to store. Downloading all the data from the services locally is not a pragmatic solution. In the 2009 ACM Workshop on Cloud Computing Security, Proofs of Retrievability (POR) was presented, which is defined as “A compact proof by a file system to a client that a target file, F is intact, and the client can fully recover it. Ideally, the client submits requests for the target data (In the form of encoded data blocks) and verifies server responses.

4. How to provide better than heuristics-based and best-effort cloud QoS?

The solution here would be to build cloud services that are controllable, i.e., we should be able to observe the behavior of the services online and monitor QoS. The Observed results can be analyzed using benchmarking tools based on realistic workloads, data loads, fault loads, and attack loads. Furthermore, their impact on the actual performance, dependability, and security of the service can be measured.

The underlying idea is to provide more power to the users to understand and operate through their data. The design proposal which we will be studying is on the grounds of providing more power to the sender but on the protocol level rather than on a generic level such as this.

2. The QUIC Transport Protocol: Design and Internet-Scale Deployment:

Introduction:

In this section, We will be discussing about the protocol in consideration here- QUIC (QUIC UDP Internet Connections). This figure below depicts QUIC in the HTTPS stack.

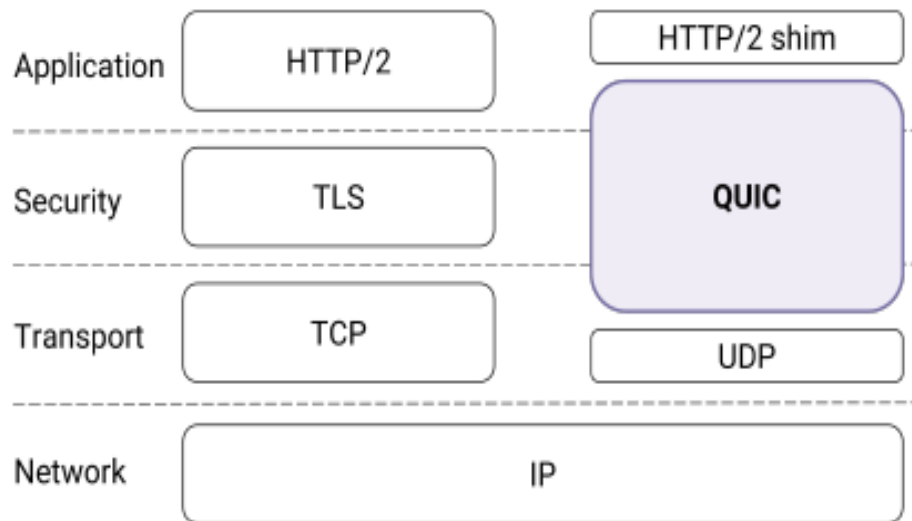


Figure 21 QUIC protocol [3]

Let's begin with, what is QUIC protocol?

According to Cloudflare, 'QUIC (Quick UDP Internet Connections) is a new encrypted-by-default Internet transport protocol, that provides a number of improvements designed to accelerate HTTP traffic as well as make it more secure, with the intended goal of eventually replacing TCP and TLS on the web.'

Major advantages of QUIC:

1. Built-in performance and security

One of QUIC's profound deviations from the currently standardized TCP, is the dire need and importance of designing a transport protocol which is secure by default. How does QUIC stand true to this revelation? It does so by providing security features, like **encryption** and **authentication**, that are incorporated by a higher layer security protocol (like TLS-Transport

Layer Security)). During the connection establishment phase, the initial QUIC handshake combines the usual three-way handshake followed by the TLS, along with the TLS 1.3 handshake, that incorporates the negotiation of cryptographic parameters as well as the authentication of the endpoints. QUIC basically replaces the TLS record layer (in the header format) with its own framing format, whilst keeping the TLS handshake messages exactly the same. We will delve into the details of the QUIC architecture in the sessions to come.

Although the generic nature and the standardized security features of TCP and TLS continue to aid in the progression of the internet technology evolution, the costs associated with the layering have been quite visible with the ever-increasing latency dependence on the HTTPS stack. TCP connections are generally said to incur at least one RTT delay of connection establishment time before any application data can be forwarded. Furthermore, TLS adds two more round trips to this delay. Most of the connections on the Internet today, and certainly most of the transactions on the world wide web being short transfers, are most impacted by unnecessary handshake round trips.

So how does QUIC overcome the above-mentioned challenge? The typical QUIC handshake only takes a single RTT between the client and server to complete the connection establishment phase as opposed to two round-trips required for the TCP and TLS 1.3 handshakes combined. Given below are two figures to compare the connection establishment between TCP and QUIC [3].

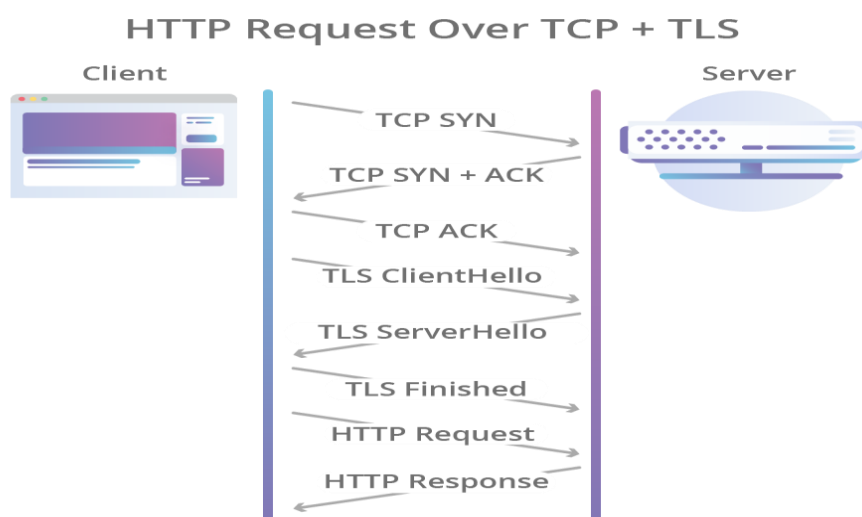


Figure 22 HTTP request over TCP+TLS (source: Quora)

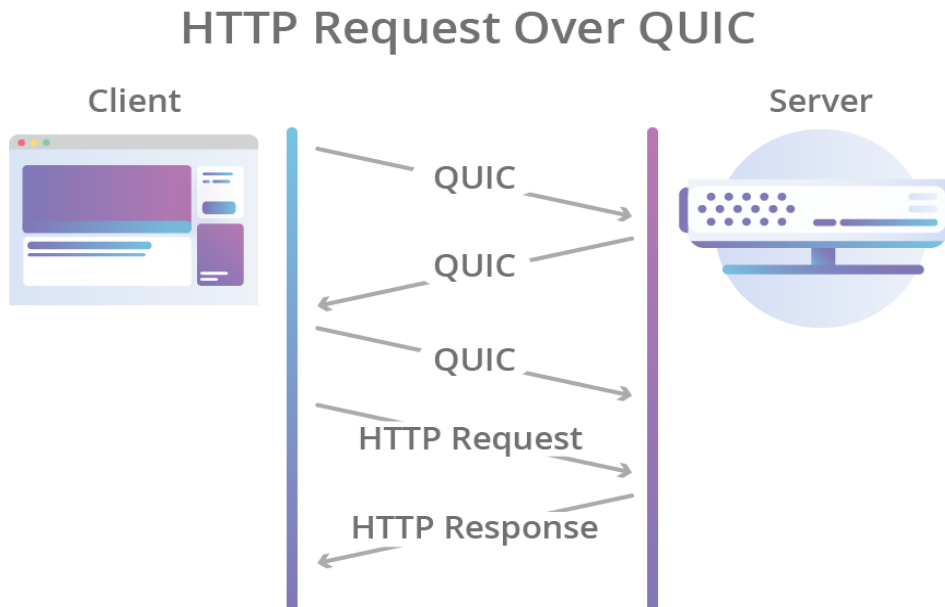


Figure 23 HTTP request over QUIC Protocol (source: Quora)

Head of line Blocking

QUIC goes a bit deeper implements multiplexing in such a way that different HTTP streams can be mapped to various QUIC transport streams whilst sharing the same QUIC connection. Hence, no additional handshakes are needed, and the congestion state is shared. QUIC streams are usually delivered independently, i.e., the loss of packets in one stream doesn't affect other streams. This can dramatically cut down the time required to, for instance, render complete web pages (with JavaScript, CSS, images, etc.), especially when crossing through highly congested networks with high packet loss rates.

In order to reduce overhead costs of using multiple TCP connections and latency, HTTP/1.1 suggests that we limit the number of connections initiated by a client to any server [3].

Furthermore, in order to reduce transaction latency, HTTP/2 multiplexes objects, and recommends that we use a single TCP connection to any server. TCP's byte stream abstraction, however,

prevents applications from controlling the framing of their communications and enforces what is called as a "latency tax" on application frames that depend upon retransmissions of previously lost TCP segments for their delivery. In general, the incorporation of transport modifications for the web requires the transport stack in server and/or client OSes, changes to web servers and clients, and often to some intervening middleboxes. Deploying changes to all three components requires incentivizing and coordinating between OS vendors, application developers, the network operators, and the middlebox vendors. **QUIC builds transport functions atop UDP and encrypts transport headers, thereby avoiding any sort of dependence on network operators and vendors.**

2. Security

As we know, QUIC is an encrypted transport protocol: packets are encrypted and authenticated; therefore, limiting ossification and preventing any kind of modification of the protocol by middleboxes. QUIC uses known server credentials on repeat connections and removes any redundant handshake-overhead at the network stack to reduce the handshake latency incurred for most connections. Additionally, it also makes use of a cryptographic handshake. By using lightweight data-structuring abstraction streams, QUIC eliminates head-of-line blocking delays. In order to restrict the blocking of streams to only the data in the packet that faces any kind of packet loss, the abstraction streams are multiplexed within a single connection [3].

Applications of QUIC today:

On the server-side, QUIC is deployed at Google's front-end servers, which simultaneously handle a huge network traffic- billions of requests a day from various web browsers and mobile applications spanning across a wide range of services. On the client-side, QUIC is deployed in Google Chrome, in the very popular mobile video streaming YouTube application, and in the Google Search app on Android. QUIC's latency advantages have been studied, and the conclusions drawn give us the following statistics- QUIC is known to reduce the Google Search latency responses by 8.0% for desktop users and by 3.6% for mobile users. Furthermore, it reduces the YouTube rebuffer rates by 18.0% for desktop users and 15.3% for mobile users. It currently accounts for over 30% of Google's total egress traffic in bytes and, consequently, an estimated 7%

of global Internet traffic[3][1]. The latest technology to implement this protocol is **THE CLOUD-Quic.cloud** (We will discuss more about it in the coming sections).

Design and Implementation:

The main goals of this protocol- Deployability, Security, Reduction in handshake, and head of line blocking delays. Let us take a peek into the basic steps involved in the working of the QUIC protocol[3]:

1. Connection Establishment Phase:

QUIC relies on a combined transport and cryptographic handshake for regulating a secure transport connection. For a successful handshake, a client usually caches information about the origin. On subsequent connections being made to the same origin, the client can thereby enforce an encrypted connection with no additional RTTs, and the application data can be sent immediately following the client handshake packet without waiting for a reply from the server. QUIC provides a reliable and dedicated stream (the streams are described in Figure 24) for performing the cryptographic handshake.

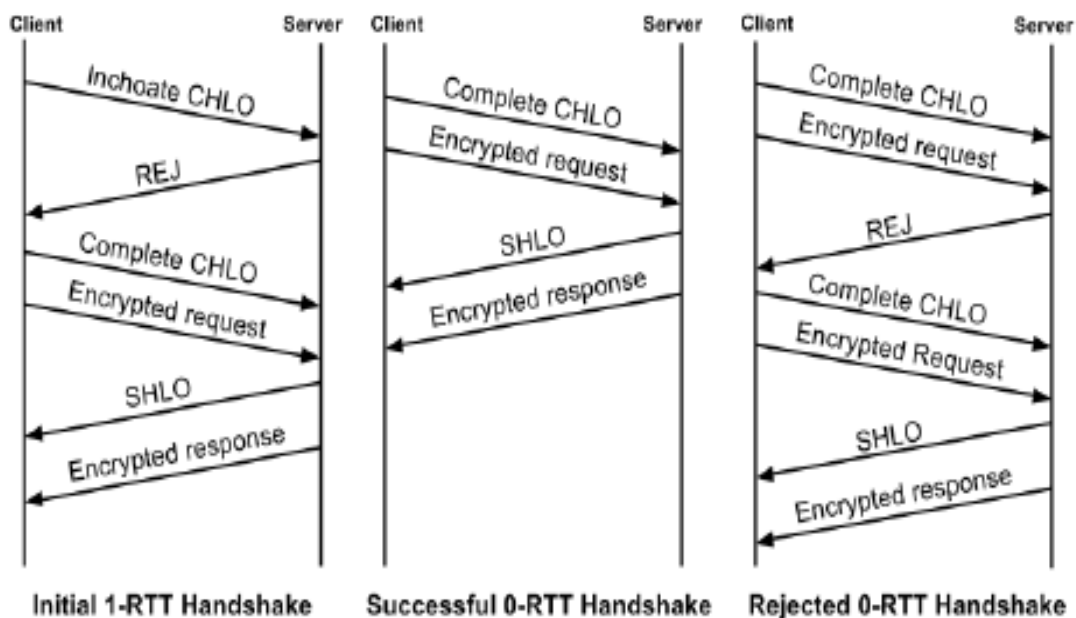


Figure 24 Timeline of QUIC's initial 1-RTT handshake, a subsequent successful 0-RTT handshake, and a failed 0-RTT handshake.

Initial handshake: In the beginning, the client has no absolutely no information about the server, and hence, before a handshake is initiated, the client sends across an inchoate client hello (CHLO) message to the server in order to evoke a reject (REJ) message. The REJ message from the server contains:

- (i) a server configuration that includes the server's Diffie-Hellman public value,
- (ii) a certificate chain to authenticate the server,
- (iii) Using the private key from the leaf certificate of the chain, a signature of the server config is obtained

and (v) a source-address token: an authenticated-encryption block that consists of the client's public IP address and the server's timestamp. In the later handshakes, the client then sends this token back to the server, thereby demonstrating ownership of its IP address. Upon receiving the server configuration, the client authenticates the configuration by verifying the whole chain and sends a complete CHLO that contains the client's transitory Diffie-Hellman public value[3].

The Final (and repeat) handshake phase: All keys for a connection are calculated using the Diffie-Hellman process. On sending across a complete CHLO, the client is now in possession of the initial keys needed for the connection since with that in hand; it can effectively determine the shared value from its own ephemeral Diffie-Hellman private key and the server's long-term Diffie- Hellman public value, Right after this, the client can start sending application data to the server. The client can send the data by encrypting it with its initial keys ahead of waiting for the reply from the server if it wishes to achieve 0-RTT latency for data. Upon a successful handshake, the server returns a Server Hello (SHLO) message. This message contains the server's transitory Diffie-Hellman public value and is encrypted using the initial keys. With the peer's transitory public value in hand, both sides can now determine the forward-secure keys or final keys or for the connection. After sending the SHLO message, the server immediately starts sending packets by encrypting it with the forward-secure keys.

Therefore, there are two levels of secrecy provided by QUIC's cryptography: the initial client data is encrypted using initial keys, and the subsequent server data and client data use the forward-secure keys for encryption. Hence, from the process, we understand how both, the initial keys and the forward-secure keys aid in providing greater protection. In case of a repeat connection to the same origin, the client uses the cached server config and source-address token to start the connection with a complete CHLO. The client, without having the need to wait for the server's response, can now send data encrypted with the initial keys to the server. What happens if eventually the server config or the source address token expires, or the server ends up changing the certificates resulting in a handshake failure? Well, in such cases, the server simply replies with a REJ message, like it replies when it receives its first inchoate CHLO from the client, and the handshaking process proceeds from there.

Version Negotiation: The process of Version Negotiation is performed by the QUIC clients and servers during the connection establishment phase to avoid unnecessary delays. However, the version negotiation packets are not protected like the other QUIC packets. The whole process is pretty simple:

- The QUIC Client sends the very first packet of the connection housing the 'version' that it proposes to use. In fact, it encodes the rest of the handshake using the same proposed version.
- Now, what if the client and server do not speak the same version? In that case, the server sends back a version negotiation packet to the client enlisting all the supported versions, thereby causing one RTT delay before connection establishment. This latency can be eliminated if both the client and the server speak the same version, thus saving time that would be required by the client to deploy the newer versions.
- In order to prevent downgrade attacks, the list of versions supported by the server and the initial version requested by the client are both fed into the key-derivation function (KDF) at both the client and the server while the final keys are generated [3].

2. Stream Multiplexing:

QUIC streams are a lightweight abstraction that provides a reliable bidirectional byte stream. Streams can be used for framing application messages of arbitrary size. A size of up to 264

bytes, which are extremely lightweight, can be easily transferred on a single stream. Due to its lightweight property, a new stream can reasonably be used for each one when sending a series of small messages. Streams are identified by stream IDs. The allotment arrangement is very simple- even IDs for server-initiated streams and odd IDs for client-initiated streams in order to avoid collisions. Stream creation usually constant while sending the first bytes on an unused stream. The stream closing is indicated to the peer on the other end by setting a "FIN" bit on the last stream frame. If the data on a stream is determined by either the sender or the receiver that it is no longer required, then the stream can be revoked/eliminated without having to tear down the entire QUIC connection. Although, streams are said to be reliable abstractions, QUIC does not retransmit data for a stream that has been revoked[3].

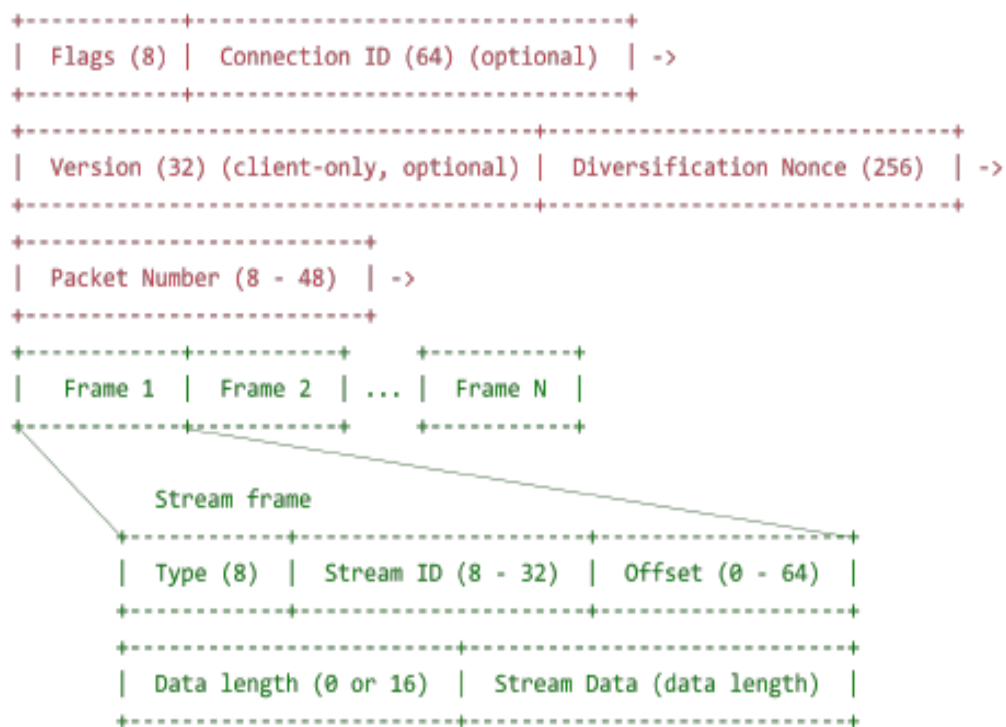


Figure 25 Structure of a QUIC packet, as of version 35 of Google's QUIC implementation. Red is the authenticated but unencrypted public header; green indicates the encrypted body.

A QUIC packet is composed of a common header followed by one or more frames, as shown in Figure 9. QUIC stream multiplexing is enforced by encapsulating the stream data in one or more stream frames. A single QUIC packet can also carry stream frames from multiple streams.

3. Authentication and Encryption phase

QUIC packets are mostly encrypted and fully authenticated, barring a few reset packets and few early handshake packets. The structure of a QUIC packet is illustrated in figure 25. The parts of the QUIC packet header outside the cover of encryption are required either for routing or for decrypting the packet: Flags, Connection ID, Version Number, Diversification Nonce, and Packet Number. Flags encode the presence of the Packet Number field length and the connection ID field. The Connection ID serves routing and identification purposes; it is used by load balancers to direct the connection's traffic to the right server and by the server to locate the connection state. The diversification nonce is generated by the server and sent to the client in the SHLO packet in order to add entropy into key generation. The diversification nonce fields and the version number and are only present in early packets [3].

4. Loss Recovery in QUIC

How does QUIC deal with loss recovery better than the TCP? To begin with, let us define the “retransmission ambiguity” problem in TCP. TCP sequence numbers represent the order in which bytes are to be delivered at the receiver, thereby ensuring reliability. But the retransmitted TCP segment carries the same sequence numbers as the original packet, which causes the ambiguity problem. The receiver of a TCP ACK cannot really determine whether the ACK was sent for a retransmission of the original transmission. In addition, the loss of a retransmitted segment is usually detected via an expensive timeout. How does QUIC overcome the ambiguity problem? Every QUIC packet carries a new and unique packet number, including the packets carrying the retransmitted data. This arrangement does away with the need for a separate mechanism to distinguish the ACK of an original transmission from that of a retransmission, thus overcoming the TCP's retransmission ambiguity problem. QUIC employs the packet number to represent an explicit time-ordering, which enables much more accurate and simpler detection of lost packets as compared to the way it is dealt with in TCP. The delay between receiving the packet and its corresponding acknowledgment being sent is explicitly encoded by the QUIC Ack. Together with monotonically increasing packet numbers, this allows for precise network round-trip time (RTT) estimation, which aids in loss detection. Accurate RTT estimation can also help the delay-sensing congestion controllers such as BBR and PCC with the required information. QUIC's acknowledgments support up to 256 ACK

blocks, making QUIC more resilient to reordering and loss than TCP with SACK. Furthermore, QUIC can keep several more bytes on the wire than the TCP in the presence of reordering or loss[3].

5. Flow Control in QUIC

Flow control generally limits the buffer size that the receiver must maintain when an application reads data slowly from QUIC's receive buffers. Consume The entire connection's receive buffer can be consumed by a slowly draining system, thereby blocking the sender from sending data on to other streams. QUIC mitigates the potential of head-of-line blocking by setting a buffer limit on what a single stream can consume. QUIC thereby employs two types of flow-control, namely, a **stream-level flow control**- to set a buffer limit for the sender to consume on any given stream and a **connection-level flow control**- to set the aggregate buffer limit that a sender can consume at the receiver across all streams. Similar to HTTP/2, QUIC employs credit-based flow control. This means the absolute byte offset is advertised within each stream up to which the receiver is ready to receive the data. In order to increase the advertised offset limit for a particular stream, the receiver periodically sends window update frames. This allows the peer to send more data on the same stream [3].

6. Congestion Control

The QUIC protocol does not rely on a specific congestion control algorithm, and the suggested implementation has a pluggable interface to allow experimentation. In the subsequent deployment, TCP and QUIC both use Cubic as the congestion controller, with one difference worth noting. For video playback on both desktop and mobile devices, the non-QUIC clients use two TCP connections to the video server to video and audio data. The connections are not designated as audio or video connections; each chunk of audio and video arbitrarily uses one of the two connections. Since the audio and video streams are sent over two streams in a single QUIC connection, QUIC uses a variant of mulTCP for Cubic during the congestion avoidance phase to attain parity in flow-fairness with the use of TCP[3].

7. NAT Rebinding and Connection Migration in QUIC

QUIC connections are usually identified by a 64-bit Connection ID. The function of this Connection ID is that it helps the connections to survive any changes to the client's IP and port (For example, migration). Such changes can easily be due to NAT timeout and rebinding (which is said to be much more aggressive for UDP as compared to TCP) or by the client simply changing his/her network connectivity to a new IP address (Migration). The major advantage of using the connection ID to identify connections is to suppress the problem of NAT rebinding. However, client-initiated connection migration has a limited deployment scale as of today [3].

8. QUIC over HTTPS

A client does not know a priori whether a given server speaks QUIC. When our client makes an HTTP request to an origin for the first time, it sends the request over TLS/TCP. Our servers advertise QUIC support by including an "Alt-Svc" header in their HTTP responses. This header tells a client that connections to the origin may be attempted using QUIC. The client can now attempt to use QUIC in subsequent requests to the same origin. On a subsequent HTTP request to the same origin, the client races a QUIC and a TLS/TCP connection, but prefers the QUIC connection by delaying connecting via TLS/TCP by up to 300 ms. Whichever protocol successfully establishes a connection first ends up getting used for that request. If QUIC is blocked on the path, or if the QUIC handshake packet is larger than the path's MTU, then the QUIC handshake fails, and the client uses the fallback TLS/TCP connection.

Comparison of Handshake Latency for QUIC and TCP:

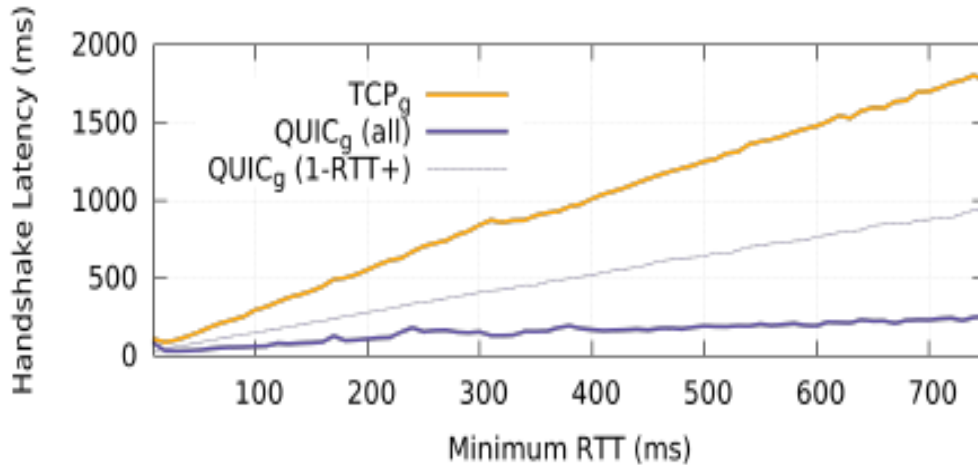


Figure 26 Comparison of handshake latency[3]

The time taken to establish a secure transport connection is called the handshake latency. In TLS/TCP, this includes the time taken for both the TCP and TLS handshakes to be completed. In this paper, they measured handshake latency at the server as the time from receiving the first TCP SYN or QUIC client hello packet to the point at which the handshake is considered complete. In the case of a QUIC 0-RTT handshake, latency is measured as 0 ms. **Figure 10** shows the impact of QUIC's 0-RTT and 1-RTT handshakes on handshake latency. With increasing RTT, average handshake latency for TCP/TLS trends upwards linearly, while QUIC stays almost flat. QUIC's handshake latency is largely insensitive to RTT due to the fixed (zero) latency cost of 0-RTT handshakes, which constitute about 88% of all QUIC handshakes. The slight increase in QUIC handshake latency with RTT is due to the remaining connections that do not successfully connect in 0-RTT. Note that even these remaining connections complete their handshake in less time than the 2- or 3-RTT TLS/TCP handshakes.

Comparison of TCP and QUIC in terms of- Mean Search Latency, Mean Video Latency, and Mean Rebuffer rate :

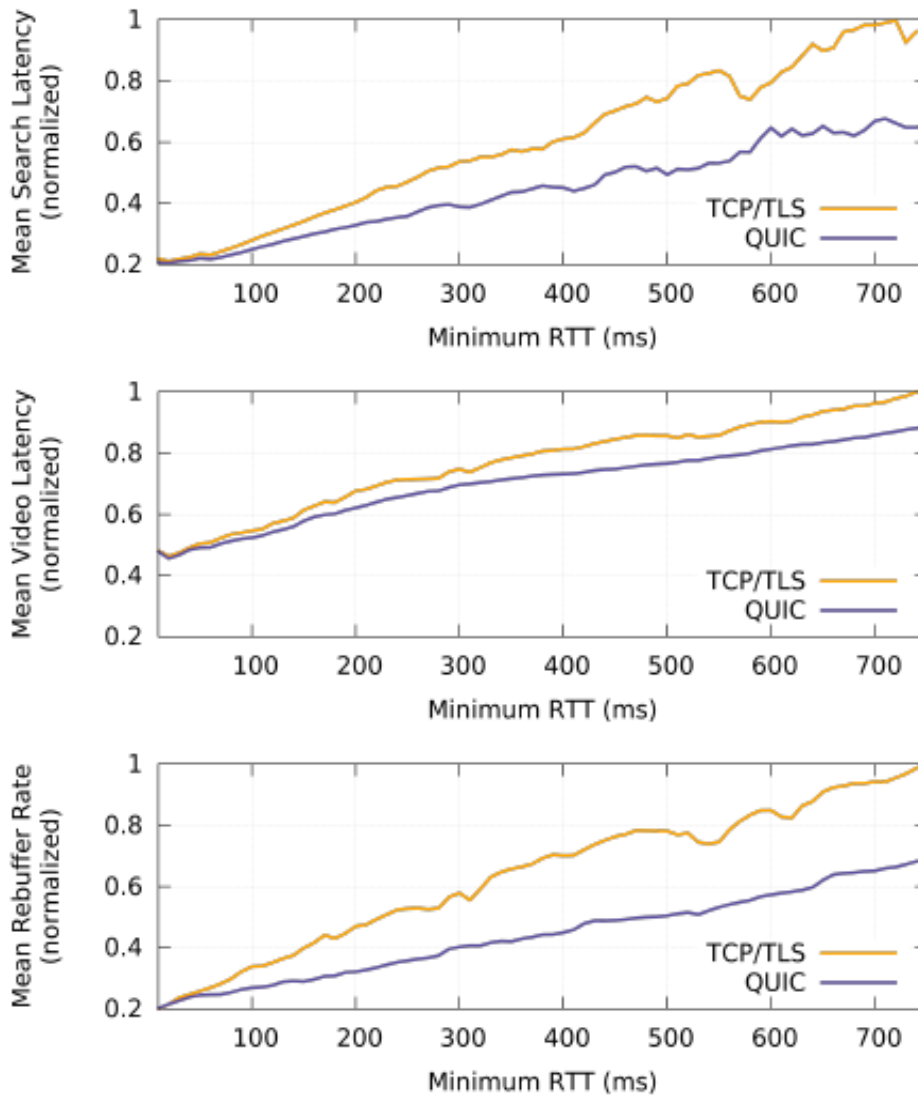


Figure 27 Comparison between TCP and QUIC[3]

The Y-axis is normalized against the maximum value in each dataset.

Server CPU Utilization

The QUIC implementation was initially written with a focus on rapid feature development and ease of debugging, not CPU efficiency. When the authors in this paper started measuring the cost of serving YouTube traffic over QUIC, they found that QUIC's server CPU-utilization

was about 3.5 times higher than TLS/TCP. The three major sources of QUIC's CPU cost were: cryptography, sending and receiving of UDP packets, and maintaining internal QUIC state. To reduce cryptographic costs, they employed a hand-optimized version of the ChaCha20 cipher favored by mobile clients. To reduce packet receive costs, they used asynchronous packet reception from the kernel via a memory-mapped application ring buffer (Linux's `PACKET_RX_RING`). Finally, to reduce the cost of maintaining state, we rewrote critical paths and data-structures to be more cache-efficient. With these optimizations, they decreased the CPU cost of serving web traffic over QUIC to approximately twice that of TLS/TCP, which has allowed us to increase the levels of QUIC traffic we serve. While QUIC will remain more costly than TLS/TCP, further reductions are possible. Specifically, a general kernel bypass seems like a promising match for a user-space transport [3].

Experiment Conducted:

The authors performed a simple experiment to choose an appropriate maximum packet size for QUIC. They performed a wide-scale reachability experiment using Chrome's experimentation framework. They tested a range of possible UDP payload sizes, from 1200 bytes up to 1500 bytes, in 5-byte increments. For each packet size, approximately 25,000 instances of Chrome would attempt to send UDP packets of that size to an echo server on our network and wait for a response. If at least one response was received, this trial counted as a reachability success, otherwise it was considered to be a failure. **Figure 13** shows the percentage of clients unable to reach the servers with packets of each tested size. The rapid increase in unreachability after As a result of the total packet size— QUIC payload combined with UDP and IP headers(1450 bytes), there is a rapid increase in unreachability, thereby exceeding the 1500-byte Ethernet MTU. Based on this data, 1350 bytes has been chosen as the default payload size for QUIC.

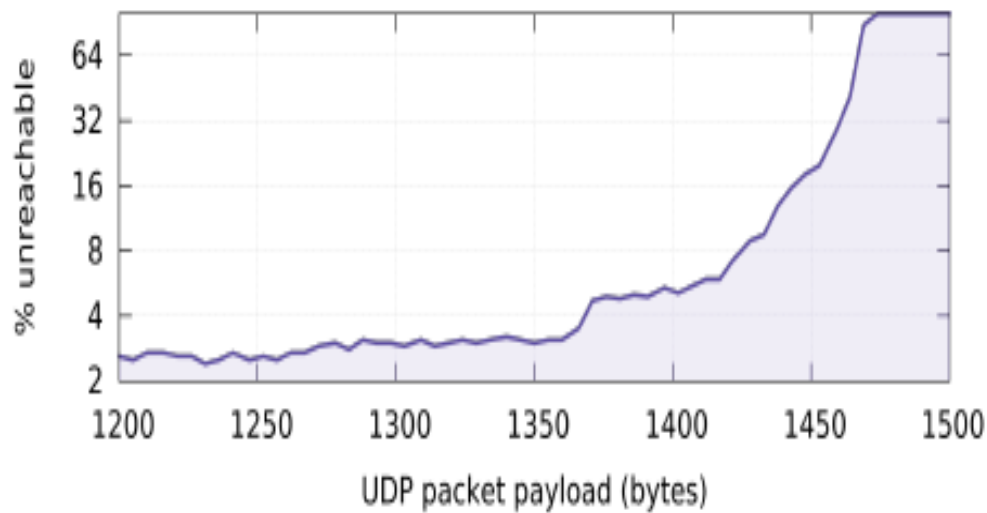


Figure 28 Unreachability of packet payloads

Some Take away points:

- The QUIC protocol combines its cryptographic and transport handshakes to minimize setup RTTs.
- It multiplexes multiple requests/responses over a single connection by providing each with its own stream, so that no response can be blocked by another.
- It encrypts and authenticates packets to avoid tampering by middleboxes and to limit the ossification of the protocol.
- It improves loss recovery by using unique packet numbers to avoid retransmission ambiguity and by using explicit signaling in ACKs for accurate RTT measurements.
- It allows connections to migrate across IP address changes by using a Connection ID to identify connections instead of the IP/port 5-tuple.
- It provides flow control to limit the amount of data buffered at a slow receiver and ensures that a single stream does not consume all the receiver's buffer by using per-stream flow control limits.

- This implementation provides a modular congestion control interface for experimenting with various controllers. Clients and servers negotiate the use of the protocol without additional latency.

3. How Secure and quick is QUIC? Provable Security and Performance Analyses.

Introduction:

This research paper introduces a security model for analyzing performance-driven protocols such as QUIC and studies and analyze the results to ideally understand where we can possibly improve. One of the most upcoming protocols is QUIC, a secure transport protocol developed by Google and implemented in Google Chrome in 2013. QUIC amalgamates ideas from TCP, TLS, and DTLS- implements congestion control comparable with TCP, provides security functionality comparable to TLS, as well as focuses on minimal RTT costs during the connection setup and in response to packet loss scenarios.

QUIC does not rely on TCP to eliminate redundant communication, and it uses a set of initial keys to establish a faster connection. QUIC has been seen as being conceptualized for mobile applications and web content delivery. It may eventually be deployed outside of Google servers. Therefore, it is highly critical to provide an accurately provable security analysis and to understand the performance guarantees when faced with an attack before it becomes more widely used.

This paper considers an attacker who is assumed to know all the servers' public keys, can, drop, intercept, disorder, or modify the contents of the packets that are exchanged and can initiate and observe the communications between honest parties. DoS attacks such as IP spoofing is also considered as a possibility. The attacker can adeptly corrupt servers, learn parties' initial and final session keys, and also learn their persisting secret states and keys. The attacker is also assumed to have a partial knowledge of the data that the parties exchange. The attackers know that the sender authentication can be achieved in a single-way, since only the servers hold public keys. The

security model discussed formally captures the different levels of security guaranteed for data encrypted under the initial and final session keys.

The attacker can actually make honest parties to agree on distinct initial keys (something that is not possible with only TLS). However, the data exchanged under either of the keys is protected. As per the session keys in TLS: which implies that if one party sets the key, the other party also sets the same key since the key is supposedly ‘good enough’ to securely exchange data. QUIC’s final session keys are no different.

Finally, the concept of forward secrecy is also considered. QUIC provides certain forward secrecy guarantees (unlike TLS-RSA, which is currently the most commonly deployed mode of TLS). This means that when the attacker corrupts a server during one time period, it does not break/breach the security of the data sent in the previous time periods. They are independent of each other. There is, however, one loophole in this arrangement. Since the initial keys that are used for the initial data exchange are derived using parameters that tend to change only once per time period, QUIC cannot effectively ensure forward secrecy guarantees against attacks that may breach the server after (although in the same period that the data was sent).

The authors then analyze the security of the cryptographic core of QUIC. They try to prove that QUIC satisfies the following security models to enhance security and reliability:

- The security model with an assumed unforgeability of the underlying signature scheme.
- The security of the underlying symmetric authenticated encryption scheme with associated data (AEAD).
- The random oracle model with a strong Diffie-Hellman assumption.

Finally, the authors delve into QUIC's latency goals in the presence of attackers[4].

Attacks under consideration:

Attack Name	Type	On-Path	Traffic Sniffing	IP Spoofing	Impact
Server Config Replay Attack	Replay	No	Yes	Yes	Connection Failure
Source-Address Token Replay Attack	Replay	No	Yes	Yes	Server DoS
Connection ID Manipulation Attack	Manipulation	Yes	No	No	Connection Failure; server load
Source-Address Token Manipulation Attack	Manipulation	Yes	No	No	Connection Failure; server load
Crypto Stream Offset Attack	Other	No	Yes	Yes	Connection Failure

Table- The attacks listed are implemented in python using scapy library[4]

Here is a brief discussion about the Replay and Manipulation Attacks[4]:

Replay Attacks:

Once a client creates a session with a particular server, an attacker could learn the source-address token value *stk* that corresponds to that particular client during their respective validity periods and the public values of that server's *safe* (used to configure the socket). The attackers could then play a double-sided attack: replay the source-address token *stk* to the server and the server's *safe* to the client thereby, misleading the other entity. However, in order to be able to launch both the attacks, the attacker will need to have access to the main communication channel.

Server Config Replay Attack

In the attempt to snoop, an attacker can replicate a server's public *scfg* to all the clients sending an initial connection request to that server whilst keeping the server in the dark about the existence of

such requests from those clients. Hence, these clients end up establishing an initial key with what they think of as the ‘server’ but without the server's knowledge. Now when they try to communicate with the server, the server will not be familiar with any of them and would thus, reject all their packets. Although, the data confidentiality is not entirely affected, the clients would waste computational resources deriving an initial key experience unnecessary and additional latencies.

Source-Address Token Replay Attack in QUIC

An attacker can pretend to a client and produce the source-address token *stk* of that particular client to the server that issued that token several times to establish any number of additional connections. This action would cause the server to believe that it is the concerned client, and it ends up establishing the final forward-secure keys and the initial keys for each connection all along without the knowledge of the client in reality. This can amplify and completely exhaust the server's computational and memory resources since any further steps in the handshake will most certainly fail. Furthermore, an attacker can generate a DoS attack on the server by creating several other connections replicating many different clients.

The irony here is that, the very parameters that aimed to minimize latency end up as the source for such attacks. These attacks are really subtle as compared to deliberately dropping QUIC handshake packets. These attacks mislead at least one party into believing that everything is going perfectly fine whilst causing a complete waste of time and resources deriving the initial keys. What can be done to resolve this? One obvious solution would be to reduce *scfg* and *stk* parameters to a one-time use, since if these parameters remain active for more than a single connection, they could be utilized by the attacker to generate multiple fake connections while seeming completely valid. The downside, however, would be, it will prohibit QUIC from achieving the 0-RTT connection establishment that stands out as one of its key features.

Packet Manipulation Attacks in QUIC

The QUIC packet fields are all not adequately protected against adversarial manipulation. If an attacker gains access to the communication channel that is used by two parties to establish a session, he can also gain access to some of the unprotected parameters such as *cid* (connection id) and the source address token *stk*. This could eventually lead the client and the server to derive

different initial keys, which would result in a failed connection establishment. For a successful attack, the adversary has to make sure that all parameters modified in this way seem consistent across all sent and received packets with respect to any single party but inconsistent from the perspective of both parties participating in the handshake.

This type of attack does not raise concerns over the confidentiality and authenticity of communication that is encrypted and authenticated under the initial key, because even though the initial keys are different, they are not known by the adversary. Note also that if parties do not agree on an initial key, they cannot establish a session key in QUIC because the final server hello packet is encrypted and authenticated under the initial key. Therefore, these attacks also do not compromise the confidentiality and authenticity of communication encrypted and authenticated under the final key.

These packet manipulation attacks are smarter than just dropping QUIC handshake packets because the client and server progress through the handshake while having a mismatched conversation, resulting in the establishment of inconsistent keys. This causes both parties to waste time and resources deriving keys and another connection state. In particular, the server performs all the processing required for a successful connection, unlike in attacks that simply drop QUIC handshake packets.

A simple strategy for mitigating this type of attack would be to have the server sign all such modifiable fields in its s reject and s hello packets (CID is unencrypted). However, this would incur the cost of computing a digital signature over all such modifiable parameters, which would, in turn, open another opportunity for a DoS attack in which the adversary, with IP spoofing, could send many initial connection requests on behalf of as many clients as it desires.

4. This final paper we discuss is by far the closest to the design proposal in the report. It is called -
“Multi-Hop Packet Tracking for Experimental Facilities.”

Introduction:

The Internet has become a complex system with increasing numbers of end-systems, applications, protocols, and types of networks. Although there is a good understanding of how data is transferred over the network, we cannot observe what happens with the data after sending and before receiving it - how packets traverse through the network and with which QoS characteristics remain unknown. Towards this objective, in this research paper, the authors have developed a multi-hop packet tracking system intended to be used in experimental facilities to begin with, such as PlanetLab, where they have made their very first tests. This paper describes the packet tracking realization and the results from the prototype implementation.

This multi-hop packet tracking method passively monitors the paths that packets take throughout the network and also records detailed hop-by-hop metrics like delay and loss. These measurements can be used, for example, for traffic engineering, for the validation of routing algorithms or traffic distribution protocols (multi-cast). Furthermore, packet tracking enables measurements of environmental conditions like cross-traffic and its influence

on the user or experimenter traffic. Tracking single packets through the network supports trace-back systems by deriving the source of malicious traffic and revealing the location of the adversary. Resource limitations usually prevent us from tracking all packets in a network. Therefore, this system uses a hash-based packet selection technique that ensures a consistent selection throughout the network while maintaining statistically desired features of the sample [6].

Architecture:

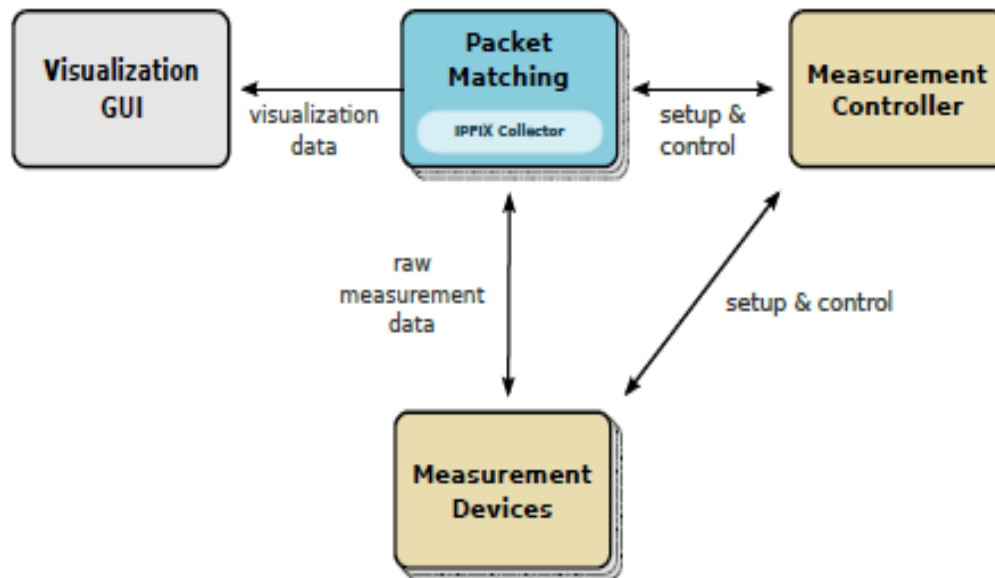


Figure 29 Multi-hop packet analysis architecture

Tracking architecture consists of:

- 1) Multiple passive measurement probes deployed in the network.
- 2) A packet matcher with an IPFIX collector that correlates the probe's measurements.
- 3) A measurement controller to co-ordinate set up and control of measurement parameters.
- 4) A visualization tool to facilitate analysis of processed data.

They use the IPFIX protocol to transfer packet tracking data from the probes to the matcher. IPFIX was primarily developed to export flow information, but also allows the reporting of per-packet information. It uses a template-based approach that assists in the definition of new information elements and also defines a standardized file format for storing measurement data (RFC5644). The probes export to the collector, at least a packet ID, and either the TTL or an arrival timestamp for each observed packet. Based on the packet ID, the packet matcher can correlate the

observations and determine the packet's direction by the TTL or timestamp. The exported timestamps can also be used for calculating the one-way delay between the observation points, which further requires that the measurement nodes' clocks are synchronized. While designing the packet tracking architecture, they have concentrated on:

1. Efficient export of measurement results. They use the IPFIX protocol and standardized Information Elements.
2. Choice of suitable packet ID generation functions. They evaluated different functions for observation correlation and decided to use the BOB hash function.
3. Reduction of measurement traffic. They use hash-based packet selection, a deterministic filter based on a hash over the packet content that synchronizes the selection of packets at different observation points. The evaluations show that hash-based packet selection can emulate random sampling when using the BOB hash function.
4. Synchronization of the sampling fractions in the network. In the case of bandwidth depletion in the network, the sampling fraction of packets should be adjusted in order to reduce the measurement traffic. To support this process, node information like bandwidth, CPU, and RAM usage are exported via IPFIX.
5. Visualization of measurement results. They use a Java visualization, which makes use of OpenStreetMap in order to visualize packet paths, their hop-to-hop characteristics, and information about the nodes[6].

Methodology:

In order to make use of the packet tracking architecture, they create a routing overlay so that the hosts also work as intermediate routers, and the authors can track packets over multiple hops. They visualize the packet path in a Java application using OpenStreetMap and a Java animation framework. An aggregate number of packets taking the same path will be visualized as a moving light dot. For the packets' paths, they used cubic splines in order to differentiate between packets that travel the same links but have different ingress and egress nodes. They use different layers to visualize node properties (CPU, RAM, sampling fraction) and link characteristics (delay). Popups over the nodes and links show a graphical representation of the data in a time window.

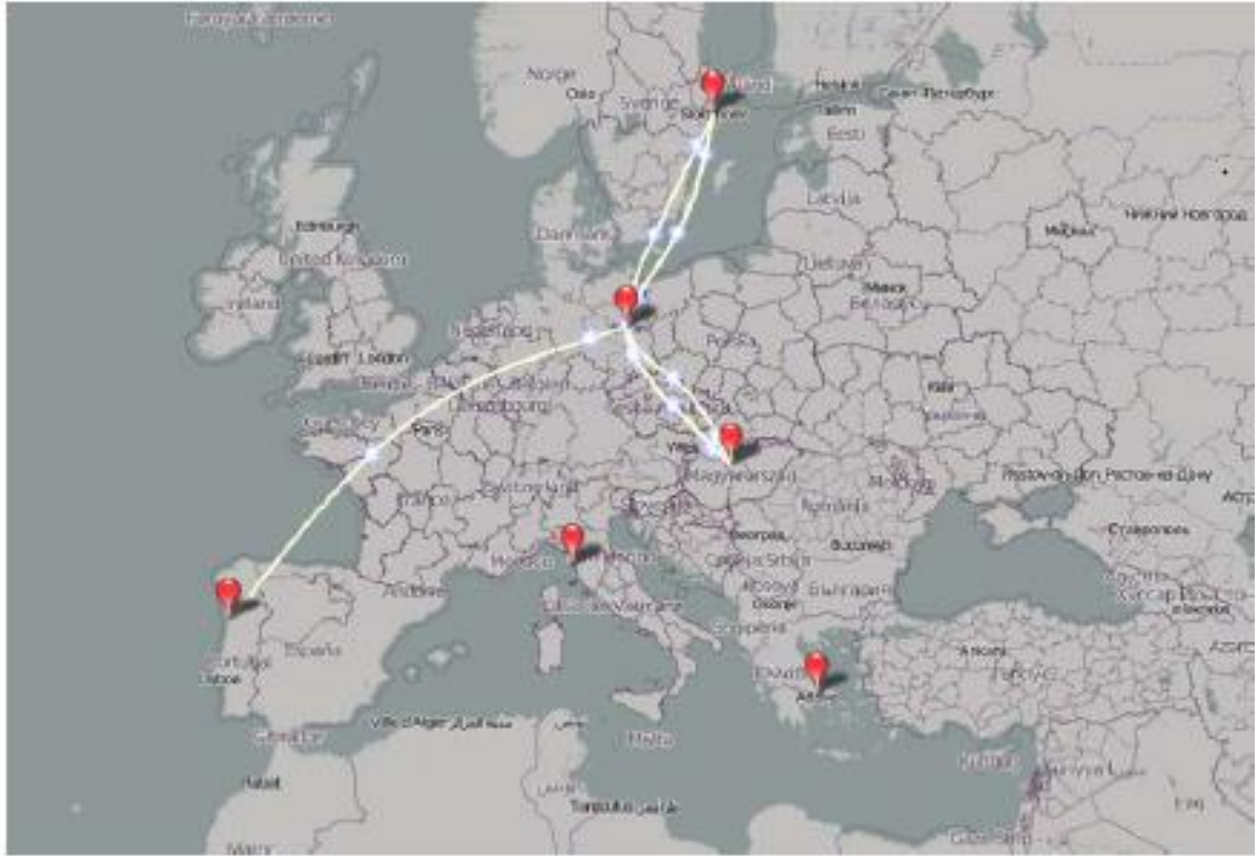


Figure 30 Snapshot of the Demonstrator

1. impd4e - a small open-source measurement probe intended for embedded systems
2. OpenMP - an open-source measurement platform including probes and measurement controllers
3. libIPFIX - an open-source C-library for exporting measurement results via the IPFIX standard
4. Packet Tracking Visualization - a Java visualization tool based on OpenStreetMap and several other open-source projects[6].

Discussion:

This is a very experimental research paper that aims to provide a holistic view of the network into consideration. A holistic view will aid in tracking packets throughout the network. This approach uses a GPS-like tracking method, thus aiding in creating a map (See Figure). One of the main disadvantages I

observe with this idea would be the restriction is the selection of packets. A hash-based selection algorithm is used in order to selectively track the packets, whereas it is important to have the tracking ability for all the packets in the network in order to increase security, accuracy, and latency. In the further sections, we shall discuss an extension of one such system that enables us to track all the packets, thereby erasing the need for acknowledgments.

QUIC Application in Cloud:

QUIC.cloud

What is it, and why is it faster?

According to their website, QUIC.cloud, by LiteSpeed Technologies, is the very first Content Delivery Network (CDN) (which is LSCache enabled) with the underlying ability to cache dynamic WordPress pages. It is currently in the beta version and is available worldwide free of charge (up to a limit- 20GB). Using QUIC as the transport protocol, QUIC.cloud aims to make websites much faster and more secure than the ones in existence today.

They strive to reduce transmission time to all users irrespective of their location. The website is potentially located across the globe from its users but with QUIC. Cloud, the content is completely cached on servers all around the world. When a visitor requests content directly from the website, a copy is served from the server location in the visitor's vicinity. By transmitting both the static and dynamic content traffic from the closest server, instead of requesting it all the way from the main server, QUIC.cloud dramatically speeds up the site. This is one of the best options for users with a slow network connection, packet loss, and high latency.

All web servers, including Apache, Nginx, openLiteSpeed, etc., are compatible with QUIC. Cloud.QUIC. Cloud acts as a CDN only and does not take the place of any web server[13].

In order to deploy this service to our systems, there are only two requirements:

1. QUIC requires HTTPS Connection to work, and most of the users generally have SSL installed automatically on their site by the system. Even if your website does not provide a private SSL/TLS certificate, it will be served using the standard HTTP protocol.
2. Either of the two browsers- Google Chrome or Opera since these are the only two browsers that support QUIC as of today.


Anti-DDoS features:

If you enable the RECAPTCHA protection inside of the QUIC. Cloud panel for the domain, they will detect if there is a high influx of connections going to your domain. If so, a verification is carried out using RECAPTCHA, which ideally helps protect against Layer 7 attacks. They also have many security-related features working behind the scenes to protect against DDoS attacks, such as blocking down bad


IPs and preventing WordPress brute force attacks [13].

With more such QUIC based applications being used in the Cloud, the need to tighten the security and fill in the gaps becomes top priority. Let us discuss about one such design proposal in the next session. Here are a few screenshots of the working of QUIC.cloud:

1. Generating the API key:

 Your profile details

Email: **shweta3@ualberta.ca** User API Key: **GqVbC2oVRGUwwhkb**

 Domains

Domain	Status
[1] sms.com	<div>CNAME not verified</div> <div>Active</div> <div>Static Cache</div> <div>Frontend SSL</div> <div>Backend SSL</div>

2. To make sure, our connections are secure, QUIC.cloud gives us two options. If the Frontend Force HTTPS is set to OFF, both request types are forwarded to the backend server (depending upon how Connection type to origin is configured).

Connection Type to Origin:

MATCH

HTTP only

When QUIC.cloud receives a request:

- **MATCH** - Connect to origin server using the same type of connection used in the frontend (HTTP will continue as HTTP, HTTPS will continue as HTTPS).
- **HTTP ONLY** - Connect to origin server via HTTP (WARNING: This will downgrade any HTTPS connections to HTTP).

Frontend Force HTTPS:

OFF ON

- **ON** will redirect HTTP requests to HTTPS via a 301 header.

3. Since certain browsers do not support QUIC, this option has been given. This is just to specify priority. In case the QUIC option is not available, it will use the default option. However, using a QUIC connection will ensure greater transmission speeds.

Enable QUIC Backend:

OFF ON

- **ON** will process the backend's response headers for an Alt-SVC header. If a supported QUIC version is available, QUIC.cloud will attempt to connect via QUIC.
You can test if your backend supports QUIC using HTTP3Check. Use the Advanced Search option and specify your domain and backend IP.
- **NOTE:** This will only take effect if the Connection Type to Origin is MATCH.
- **NOTE:** This does not affect the connection from client browsers to QUIC.cloud.

4. The option below is given to strengthen security and employ CDN.

Enable CDN:

OFF ON

Disabling this will bypass QUIC.cloud CDN and expose your IP to public.

Currently your IP is exposed to public, your domain is bypassing QUIC.cloud CDN.

Enable Static Cache:

OFF ON

Enable this to cache static files.

You are not enabling static files cache currently.

After setting up all the security requirements above according to the services needed, we go on to create a ReCaptcha site key for both the client and the server's side. Here, we also need to select the version of reCAPTCHA. I selected V2 in this case. These two keys have been generated for the domain created using QUIC.cloud:

6LdrzegUAAAAAJR2cwHQym5liXcjUFpUbYW_uDZX- reCAPTCHA client

6LdrzegUAAAAACwdPaG0y9h-AOV1XZ1KDVGlC7Eu- reCAPTCHA server

reCAPTCHA Type: ☒ Checkbox ☐ Invisible

Only support V2 now.

reCAPTCHA Connection Limit:

Maximum value is 10,000.

5. Finally, the domain name and my domain IP address are created with a monthly traffic limit of 20GB.

Domain Name: **sms.com**

Monthly Traffic Limit: **20,000** Mb

Remain Traffic of This Month: **20,000** Mb

Domain IP: **65.207.67.138** [Change](#)

Update CNAME Records:

Please change your CNAME to **c15723.tier1.quic.cloud**

Required. Verify correct CNAME setup to enable QUIC.cloud CDN.

Figure 31 Snapshot of the Domain created on QUIC.cloud

Observations:

After having used a well-known cloud platform like Amazon Web Services (AWS) and experimenting with QUIC.cloud, these are certain observations made:

- It uses a very simple, straightforward, and user-friendly approach. It makes the whole process of creating a domain easy and less time-consuming. Most of the important modules like the API key, reCAPTCHA, security requirements can be set using a click of a button.
- It exhibits faster transmission rates as compared to the conventional providers.
- Although in the beta stage, it employs some security modules like IP whitelisting and blacklisting. This Anti-DDOS feature will be very useful for resource and service management in organizations (Please see the screenshot below).

Anti DDOS OFF ON

Recommended to ALWAYS keep "ON". Make sure it is enabled if your site is under attack. Visitors will get a reCAPTCHA before proceeding.

Anti DDOS Whitelist:
The IPs listed will not meet the recaptcha.

68.150.220.130

Anti DDOS Blacklist:
The IPs listed will be blocked directly.

80.187.53.167

Save IP lists

Figure 32 Snapshot of the Anti-DDOS features of the QUIC.cloud

- Another feature is- Several domains can be created and left to run at a time.
- There are still miles to go for QUIC.cloud to incorporate several other features since the features are quite limited at the moment.
- QUIC.cloud requires that your DNS maps your domain to a domain provided during setup which means, the DNS provider must support domain to domain mapping for the root domain. Since this is not a standard feature, if not available in our case, we must switch to a DNS provider that is capable of this feature (For Example- Cloudflare's DNS).

Theory- Track your Track:

Introduction:

Track your Track simply means, in regard to networking, a tracking facility to track the sent packets in a network. Why would one want to track the sent packet? Well, ideally, it is to make sure the packet reaches the required destination, to be able to detect any illegal reroutes, improve latency by removing the need for acknowledgments (whether it is about the packet received or about asking for a “resend”). This is analogous to one tracking their Uber ride. The more control the sender has over the sent packet, the more secure it is. Before we delve into further details of the proposed design, it is important to understand the working of the TLS (TLS 1.3) security architecture in securing the QUIC protocol. This understanding is completely based on the latest IETF document that explicates the whole security mechanism. This graph below represents the Cloud CDN throughput of a google project after enabling QUIC as recorded by Cedexis benchmarking:

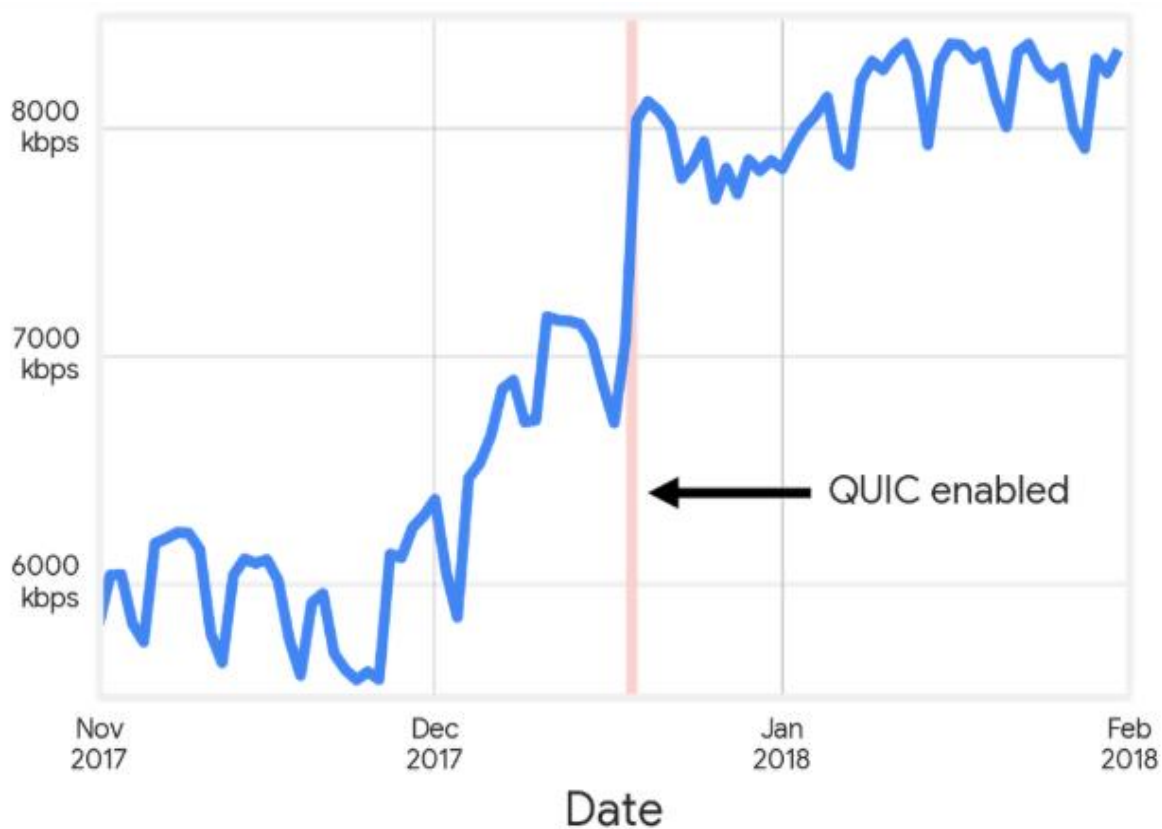


Figure 33 Graph- Cloud CDN Throughput

What is TLS?

TLS stands for Transport Layer Security. It implements two endpoints with a way to establish a means of communication over an untrusted medium (that is, the Internet) that makes sure that messages that are exchanged cannot be observed, forged, or modified.

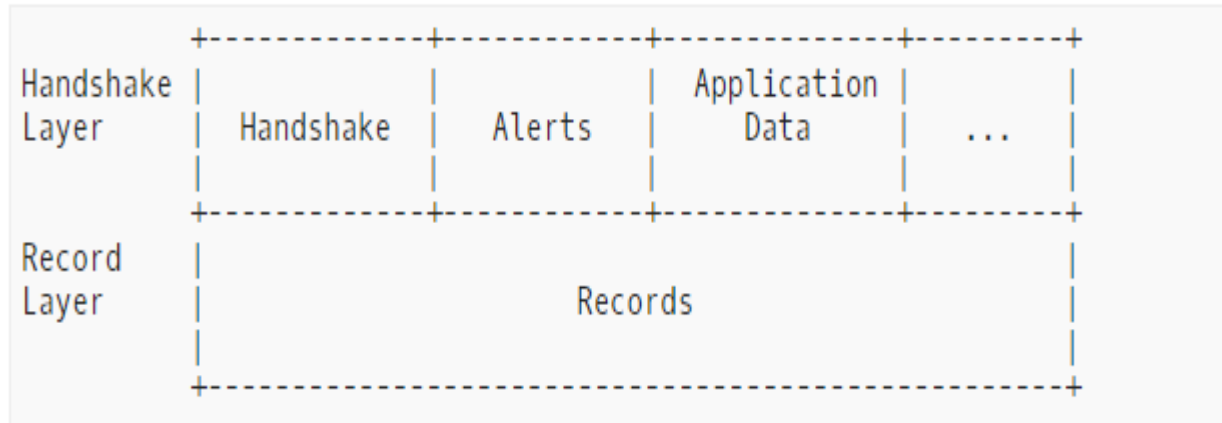


Figure 34 Structure of TLS

Every handshake layer message (e.g., Handshake, Application Data, and Alerts) is imported as a series of typed TLS records by the TLS Record layer. Individual Records are cryptographically protected, after which it is transmitted over a reliable transport (like TCP) that provides packet guaranteed delivery and sequencing.

The TLS authenticated key exchange process occurs between two endpoints: the **client** and the **server**. The client commences the exchange of keys, and the server responds. If the key exchange is completed successfully, both the client and server will come to terms on a secret. The TLS supports both Pre-Shared Key (PSK) and the Diffie-Hellman key over either finite fields or elliptic curves ((EC)DHE) key exchanges.

After performing the TLS handshake, the client usually learns and authenticates an identity for the server, and the server is optionally able to authenticate and learn about the client's identity.

The TLS key exchange is usually less susceptible to tampering by attackers, and thus produces shared secrets that may not be regulated by either participating peers.

TLS implements two very basic handshake modes of interest to QUIC[15]:[\[1\]](#)

- One full RTT handshake (1-RTT) in which the client is effective in being able to send the application data after one whole round trip, and the server is able to immediately respond right after receiving the first handshake message from the client.
- A 0-RTT handshake in which the client makes use of the information it has gained about the server in order to dispatch application data immediately. This application data can be easily replayed by an attacker. Hence, it should not carry any kind of self-contained trigger for any ineligible action. Below is an example of how a TLS Handshake looks like with a zero round trip (0-RTT).



The Application Data is usually protected using a number of encryption levels, namely[15]:

- Initial Keys
- Early Data (0-RTT) Keys
- Handshake Keys
- Application Data (1-RTT) Key

What is QUIC?

QUIC stands for Quick UDP Connections. Google Chromium describes QUIC being nothing but, TCP+TLS+HTTP/2 implemented on UDP.

The key advantages of the protocol include:

- Connection Establishment Latency
- No OS requirements and fast-evolving
- Forward Error Correction
- Connection Migration
- Improved Congestion Control
- Multiplexing without Head of Line Blocking

When we talk about reduced **connection establishment latency**, it does so because, the first time QUIC connects to a server, the client performs a 1- roundtrip handshake wherein it acquires the necessary information to complete the handshake. The client sends an empty client hello for which the server responds with the information the client requires to make forward progress (Including source address token and the server certificate). The second time the client sends an empty hello packet to the server, it uses the cached credentials from the previous connection to send encrypted requests, thereby reducing the connection establishment time.

When we talk about **Congestion Control**, QUIC provides pluggable Congestion Control. Each packet (Both retransmitted and the original one) carries a sequence number, which is generally new. This usually allows a QUIC sender to distinguish ACKs (Acknowledgments) for retransmissions from the ACKs for original transmissions and thereby avoid the TCP's retransmission ambiguity problem. QUIC Acknowledgements also definitively carry the delay between the receipt of a packet and its acknowledgment being sent, all together with the monotonically increasing sequence numbers. This kind of arrangement allows for an accurate round trip calculation. Additionally, QUIC's ACK frames give a support of up to 256 NACK ranges, so QUIC is more generally resilient towards reordering of packets than TCP (with SACK) is. Hence, it is also able to place more bytes on the wire when a reordering or loss takes place. The client and the server now have a more precise picture of the number of packets received by the

peer.

QUIC is designed from scratch for exercising **multiplexing**. The packets that are carrying data for an individual stream generally only impact that specific stream even if they are lost. Each frame of the stream can be immediately stimulated to that corresponding stream on arrival, so streams without loss can continue to be reassembled and make forward progress in the application.

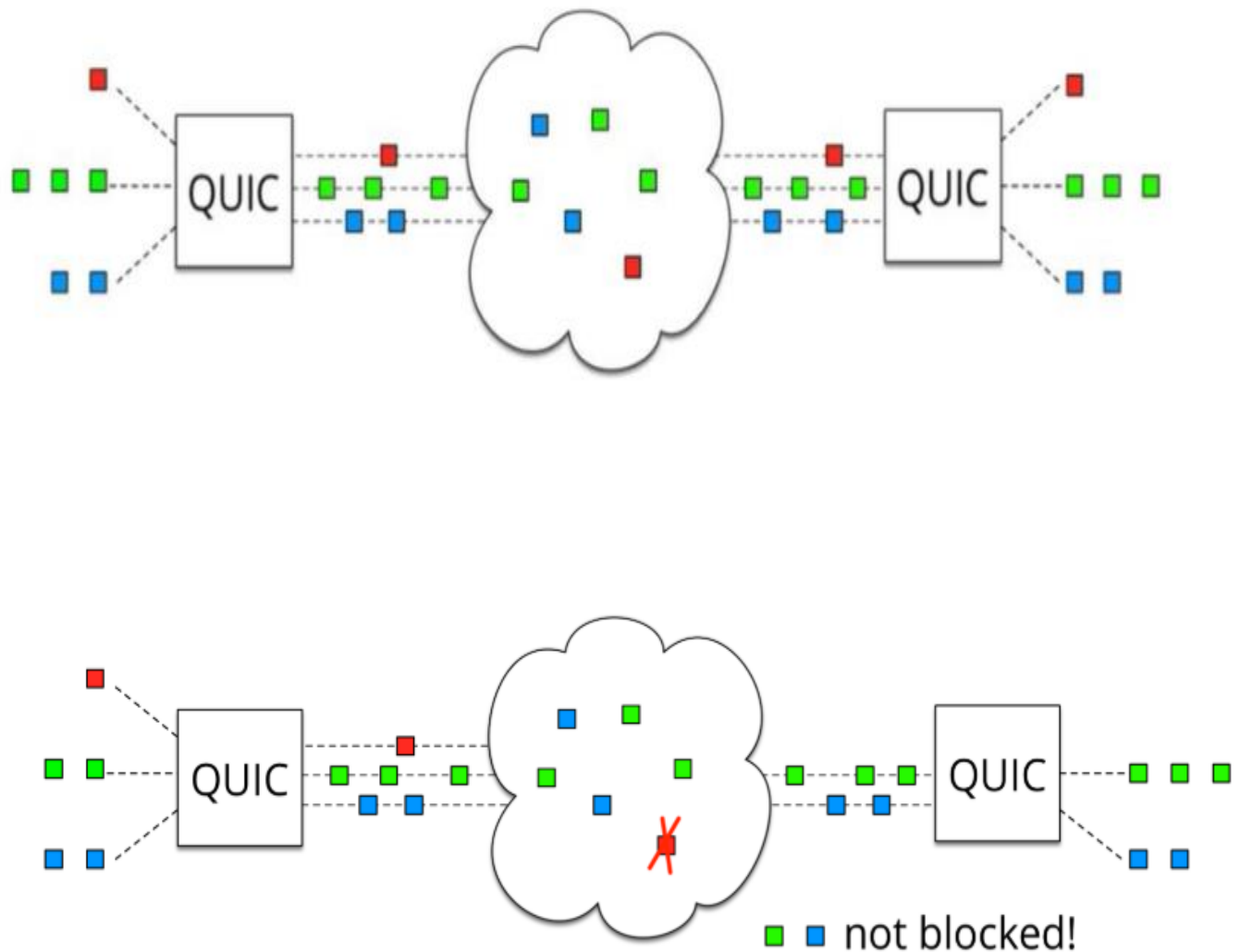


Figure 35 Source- Google Developers Live- QUIC (Slide 14)

When we talk about **Forward Error Correction**, in order to recover from lost packets without waiting for a retransmission, QUIC can complement a group of packets with an FEC packet. The FEC packet contains sent with what is called as the FEC group contains the parity of all the packets in that group. Even if one of the many packets in the group is corrupted/lost, the FEC packet can help in recovering the contents of that packet, as well as the other packets in the group. Consider the diagram below by the Google developers live:

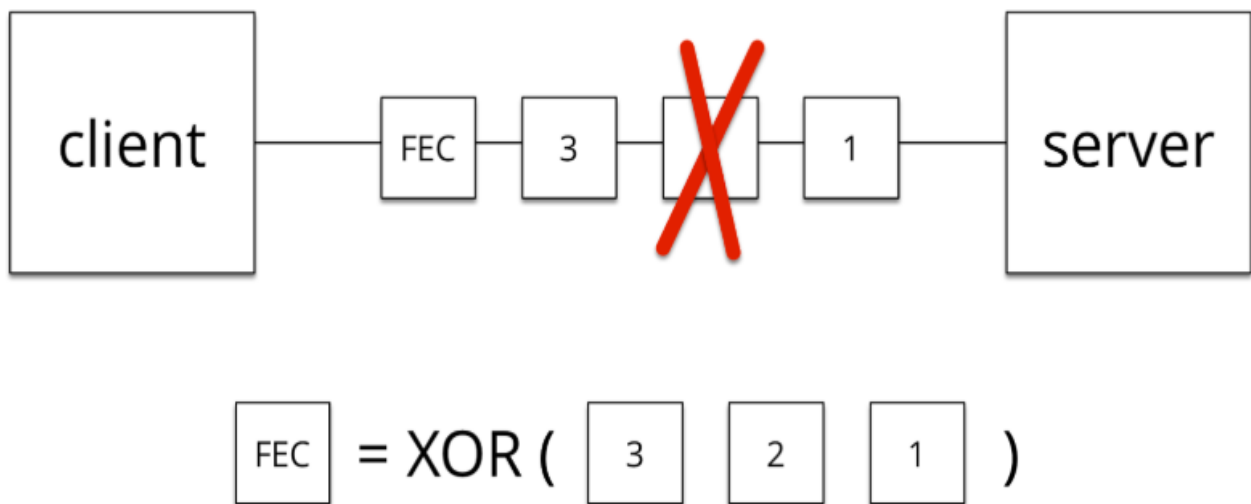


Figure 36 Forward Error Correction-Google Developers Live

Coming to **Connection Migration**- QUIC connections generally identified by a 64- bit connection ID randomly generated by the client. On the other hand, TCP connections are identified by a 4-tuple of source address, source port, destination address, and destination port. This means that if a client changes IP addresses (for example, by moving out of Wi-Fi range and switching over to cellular) or ports (if a NAT box loses and rebinds the port association), any active TCP connections are no longer valid. When a QUIC client changes IP addresses, *it can continue to use the old connection ID from the new IP address without interrupting any in-flight requests*.

QUIC Latency

QUIC considerably reduces the number of roundtrips required to set up a connection, as we have

already seen. In the recent times, Companies like Uber have started incorporating QUIC protocol in their applications. They claim to have observed a reduction of 10-30 percent in tail-end latencies for HTTP traffic at scale in their rider and driver apps as compared to using TCP for the same.

Comparing latencies between TCP, TCP+TLS, and QUIC as observed by the developers at Google:

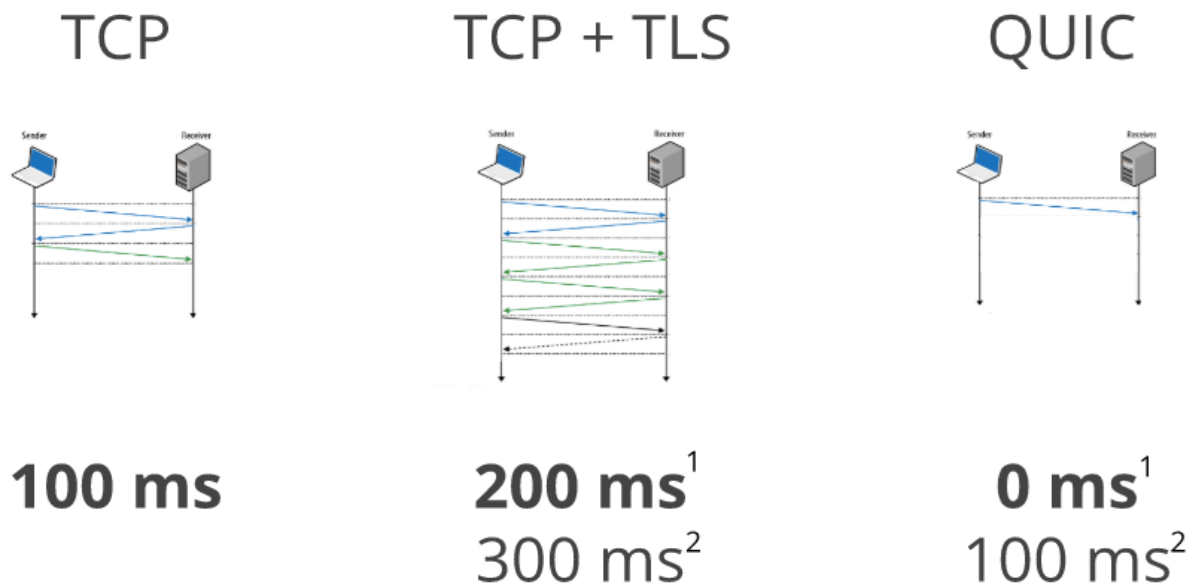


Figure 37 Comparison of latencies-Google Developers Live

As for the TCP+TLS and the QUIC protocol, the first value corresponds to the latency in case of a repeat connection, and the second value corresponds to the latency in case of a brand-new connection.

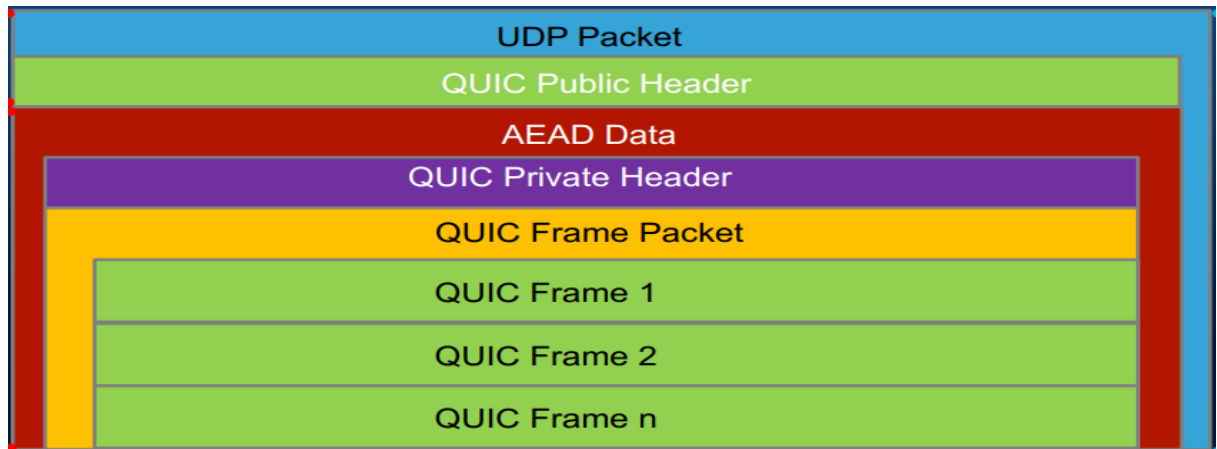


Figure 38 QUIC Packet Format (Source: Blackhat)

The UDP Packet and the QUIC Public Header is unencrypted and authenticated. The remaining part of the header is encrypted and authenticated.

This header format consists of the frames inside a QUIC frame packet, which is, in turn, found inside a QUIC packet.

We are now going to discuss how QUIC can be enabled in our browsers in order and then study how the traffic is recorded.

Below are some screenshots outlining the procedure and analyzing the traffic:

1. Since QUIC is an experimental protocol, you can find the option to enable/disable QUIC in the experiment section of the google chrome- `chrome://flags`. On e another browser that supports QUIC protocol is the Opera browser, but I stuck to Google Chrome for these results.

Search flags

Reset all to default

Experiments

80.0.3987.163

WARNING: EXPERIMENTAL FEATURES AHEAD! By enabling these features, you could lose browser data or compromise your security or privacy. Enabled features apply to all users of this browser.

Interested in cool new Chrome features? Try our [beta channel](#).

Available

Unavailable

Experimental QUIC protocol

Enable experimental QUIC protocol support. – Mac, Windows, Linux, Chrome OS, Android
[#enable-quic](#)

Enabled

Override software rendering list

Overrides the built-in software rendering list and enables GPU-acceleration on unsupported system configurations. – Mac, Windows, Linux, Chrome OS, Android

Disabled

QUIC

Reset all to default

Experiments

80.0.3987.163

Available

Unavailable

Experimental QUIC protocol

Enable experimental QUIC protocol support. – Mac, Windows, Linux, Chrome OS, Android
[#enable-quic](#)

Enabled

Default

Enabled

Disabled

2. Once QUIC is enabled, we are now moving on to observing the network. For this, you must first create a log of all the network traffic. This can be saved as a JSON file format on your local computer using an IDE like Visual Code Studio. It contains all the processes logged in order-crypto, handshaking, header protection.
3. In order to analyze the log file, we enter the Netlog viewer (This works like an inferior version of Wireshark). Wireshark is not used here because it is not very capable of demarcating QUIC and UDP packets. This Netlog viewer provides us with statistics on all the QUIC events line by line. Once you enter the Netlog viewer (<https://netlog-viewer.appspot.com/#import>), you can import the saved log file. Once this is done, you can stop capturing the network traffic.

The imported file gets logged in a tabular format like in the screenshot below:

This chrome-net-export-log.json file is broken into, and the following information is displayed- Export date, build, OS info, Command line, and Active Field trial groups. You can, however, disable the field trials by typing:

--disable-field-trial-config in the Command-line interface.

Log loaded.

Data Loaded

Export date	2020-04-11 02:30:37.638
Build	Google Chrome 80.0.3987.163 (official e7f6e071abe9328cdce4ffedac9822435fbd3656-refs/branch-heads/3987@{#1037})
OS info	Windows NT: 10.0.17763 (x86_64)
Command line	"C:\Program Files (x86)\Google\Chrome\Application\chrome.exe" --enable-audio-service-sandbox --flag-switches-begin --enable-quic --flag-switches-end --enable-audio-service-sandbox
Active Field Trial Groups	AsyncDnsInitialTimeoutMsByConnectionType:Default AutofillNoLocalSaveOnUnmaskOrUploadSuccess:FullyEnabled_WithStartsActive AutofillServerBehaviors:Control-2020-03b CacheStorageHighPriorityMatch:EnabledLaunch CacheStorageSequence:EnabledLaunch ChromeChannelStable:Enabled ClickToCallV2Sender:Default_20191026 DeprecateFtp:Preperiod_Default DetachedWindowsThrottle:Throttled DialMediaRouteProvider:EnabledLaunch DnsOverHttps:Default DnsUnresponsiveDelayMsByConnectionType:Default EnterpriseReportingInBrowser:EnabledLaunch ExpiredHistograms:ExpiredHistogramLogicEnabled HeapProfiling:Default HistoryServiceAndProfileSyncServiceUseThreadPool:Preperiod_Default ImprovedCookieControlsStudy:Default IndexedDBHighPriority:Control3 KeepaliveRequestPriority:Default LazyLoad:Default MetricsAndCrashSampling:OutOfReportingSample MirroringService:EnabledLaunch MostLikelyDesktopDeprecation:Default OffMainThreadServiceWorkerStartup:EnabledLaunch OmniboxDocumentProviderNonDogfoodExperiments:Default OmniboxMaxMatchesURLLimitLaunch:Desktop_OmniboxMaxMatchesWithURLLimit_Enabled_V2 OutOfBlinkCors:LAUNCHED_WITH_PARAMS_C PauseBrowserInitiatedHeavyTrafficForP2P:EnabledLaunch PreconnectSameOriginDesktop:Default ProactivelyThrottleLowPriorityRequests:Enabled_0_5_2G_Only ProfileMenuRevampIdentityPill:EnabledLaunch QUIC:FlagEnabled SRTPromptFieldTrial:NewCleanerUIExperiment SafeBrowsingAdPopupTrigger:Default SafeBrowsingAdRedirectTrigger:Default SimpleCacheTrailerPrefetchHint:EnabledLaunch SplitCacheByNetworkIsolationKey:Default_20200407 SyncButterWallet:EnabledLaunch SyncUSSPasswords:Launched TextFragmentAnchor:EnabledLaunch TranslateDesktopRefresh:EnabledTab UKM:Enabled_20180314 UMA-Dynamic-Uniformity-Trial:Group3 UMA-Population-Restrict:normal UMA-Uniformity-Trial-0.5-Percent-1:group_175 UMA-Uniformity-Trial-0.5-Percent-2:group_126 UMA-Uniformity-Trial-1-Percent:group_25 UMA-Uniformity-Trial-10-Percent:group_06 UMA-Uniformity-Trial-10-Percent-sanity:group_02 UMA-Uniformity-Trial-100-Percent:group_01 UMA-Uniformity-Trial-20-Percent:default UMA-Uniformity-Trial-5-Percent:group_13 UMA-Uniformity-Trial-50-Percent:default V8WasmCodeCache:EnabledLaunch

Figure 39 snapshot of the import network logs- quic

- Below is the screenshot of the QUIC options that are available for my connection. In the list, you will notice something known as Alt-Svc on QUIC errors. It is called the alternate service header, and its corresponding ALT-SVC HTTP/2 frame is not created specifically. Interestingly, it was a mechanism created for a server to tell a client that it runs the same service on the so and so host using a so and so protocol on the so and so port. A client that receives such a response is then advised to connect to that other given host in parallel in the background using the specified protocol, and if it is successful, it can switch its operations over to that instead of the initial connection. Hence, it behaves more like a substitute.

For example: Alt-Svc: h3="":60801."

The above example indicates that HTTP/3 is available on UDP port 60801 in the same hostname that was used to get this response.

Generally, for QUIC, Alt-Svc:quic= ":443"; v="32,33" where 'v' parameter is used for Alt-svc defined to carry Version Negotiation hints (see **Figure**) You can also see how the idle connection

timeout for QUIC is so much lesser than what it would be for a TCP connection. The maximum packet length is given as 1350 for this connection.

QUIC Option	Value
Supported Versions	Q046
Connection options	
Max Packet Length	1350
Idle Connection Timeout In Seconds	30
Reduced Ping Timeout In Seconds	15
Packet Reader Yield After Duration in Milliseconds	
Mark QUIC Broken When Network Blackholes	false
Do Not Mark QUIC Broken on Network Changes	false
Retry without Alt-Svc on QUIC Errors	true
Do Not Fragment	false
Allow Server Migrations	false
Migrate Sessions Early V2	false
Migrate Sessions on Network Change V2	false
Retransmittable on Wire Timeout in Milliseconds	false
Disable Bidirectional Streams	false
Race Cert Verification	false
Race Stale DNS On Connection	false
Estimate Initial RTT	false
Force Head of Line Blocking	false
Max Server Configs Stored in Properties	0
Origins To Force QUIC On	
Server Push Cancellation	false

Figure 40 Snapshot of the QUIC options

As we had discussed above, below is the screenshot for all the alternate service mappings for QUIC via port 443.

Alternate Service Mappings

Server	Alternative Service
https://play.google.com	quic :443, expires 2020-05-11 02:30:35,quic :443, expires 2020-05-11 02:30:35
https://ssl.gstatic.com	quic :443, expires 2020-05-11 02:30:35,quic :443, expires 2020-05-11 02:30:35
https://www.google.com	quic :443, expires 2020-05-11 02:29:10,quic :443, expires 2020-05-11 02:29:10
https://google.com	quic :443, expires 2020-05-11 02:28:34,quic :443, expires 2020-05-11 02:28:34
https://beacons.gcp.gvt2.com	quic :443, expires 2020-05-11 02:28:34,quic :443, expires 2020-05-11 02:28:34
https://chat-pa.clients6.google.com	quic :443, expires 2020-05-11 02:27:31,quic :443, expires 2020-05-11 02:27:31
https://adservice.google.com	quic googleads.g.doubleclick.net:443, expires 2020-05-11 02:27:21,quic :443, expires 2020-05-11 02:27:21,quic googleads.g.doubleclick.net:443, expires 2020-05-11 02:27:21,quic :443, expires 2020-05-11 02:27:21
https://encrypted-tbn0.gstatic.com	quic :443, expires 2020-05-11 02:27:20,quic :443, expires 2020-05-11 02:27:20
https://apis.google.com	quic :443, expires 2020-05-11 02:27:19,quic :443, expires 2020-05-11 02:27:19
https://ogs.google.com	quic :443, expires 2020-05-11 02:27:24,quic :443, expires 2020-05-11 02:27:24
https://www.gstatic.com	quic :443, expires 2020-05-11 02:26:10,quic :443, expires 2020-05-11 02:26:10
https://lh3.googleusercontent.com	quic :443, expires 2020-05-11 02:27:18,quic :443, expires 2020-05-11 02:27:18
https://fonts.googleapis.com	quic :443, expires 2020-05-11 02:27:14,quic :443, expires 2020-05-11 02:27:14
https://fonts.gstatic.com	quic :443, expires 2020-05-11 02:26:10,quic :443, expires 2020-05-11 02:26:10
https://clients1.google.com	quic :443, expires 2020-05-11 02:26:32,quic :443, expires 2020-05-11 02:26:32
https://ci5.googleusercontent.com	quic :443, expires 2020-05-10 21:40:20,quic :443, expires 2020-05-10 21:40:20
https://lh4.googleusercontent.com	quic :443, expires 2020-05-10 21:18:48,quic :443, expires 2020-05-10 21:18:48
https://25.client-channel.google.com	quic :443, expires 2020-05-11 02:27:02,quic :443, expires 2020-05-11 02:27:02
https://netlog-viewer.appspot.com	quic :443, expires 2020-05-11 02:26:32,quic :443, expires 2020-05-11 02:26:32
https://chromium.googlesource.com	quic :443, expires 2020-05-11 02:26:10,quic :443, expires 2020-05-11 02:26:10
https://people-pa.clients6.google.com	quic :443, expires 2020-05-11 02:25:36,quic :443, expires 2020-05-11 02:25:36
https://clients6.google.com	quic :443, expires 2020-05-11 02:25:31,quic :443, expires 2020-05-11 02:25:31
https://clients4.google.com	quic :443, expires 2020-05-11 02:14:57,quic :443, expires 2020-05-11 02:14:57

Figure 41 Snapshot of the alternate Service Mappings

- Below is the screenshot (figure 42) wherein you select the events you want to view. For instance, I have selected the QUIC_SESSION for which the logs appear on the right-hand side of the screen (screenshots attached below figure). If you observe, one of the options is QUIC_SESSION, which is described using static. This is another domain name used to reduce bandwidth usage and increase the network performance of the end-user. As the name suggests, it holds static content (offloaded JavaScript code, images, and CSS). This is used for the second QUIC_SESSION.

Figure 43 shows us the active QUIC connections made during the time we logged the network traffic. All of the traffic, as you can see, uses the port number 443. QUIC uses the traditional HTTP ports of 80 and 443, but that is the end of similarities. The supporting browsers and servers support this new protocol and thereby process its web traffic, but the network device in between cannot certainly differentiate between the application protocols and hence, switches to treating it like any generic layer 4 UDP traffic.

<input type="checkbox"/>	ID	Source Type	Description
<input type="checkbox"/>	2413	UDP_SOCKET	74.125.195.189:443
<input checked="" type="checkbox"/>	2414	QUIC_SESSION	
<input type="checkbox"/>	33766	URL_REQUEST	https://25.client-channel.google.com/client-
<input type="checkbox"/>	34388	UDP_SOCKET	172.217.3.174:443
<input type="checkbox"/>	34389	QUIC_SESSION	
<input type="checkbox"/>	34390	URL_REQUEST	https://ssl.gstatic.com/ui/v1/icons/mail/imag
<input type="checkbox"/>	34391	DISK_CACHE_ENTRY	https://ssl.gstatic.com/ui/v1/icons/mail/imag
<input type="checkbox"/>	34392	HTTP_STREAM_JOB_CONTROLLER	https://ssl.gstatic.com/ui/v1/icons/mail/imag
<input type="checkbox"/>	34393	HTTP_STREAM_JOB	https://ssl.gstatic.com/
<input type="checkbox"/>	34394	HTTP_STREAM_JOB	https://ssl.gstatic.com/
<input type="checkbox"/>	34395	QUIC_STREAM_FACTORY_JOB	
<input type="checkbox"/>	34396	HOST_RESOLVER_IMPL_JOB	ssl.gstatic.com
<input type="checkbox"/>	34397	UDP_SOCKET	172.217.3.195:443
<input type="checkbox"/>	34398	QUIC_SESSION	ssl.gstatic.com
<input type="checkbox"/>	34399	URL_REQUEST	https://play.google.com/log?format=json&h
<input type="checkbox"/>	34400	HTTP_STREAM_JOB_CONTROLLER	https://play.google.com/log?format=json&h
<input type="checkbox"/>	34401	HTTP_STREAM_JOB	https://play.google.com/
<input type="checkbox"/>	34402	HTTP_STREAM_JOB	https://play.google.com/

Figure 42 Event Selection- QUIC network logs

Host	Version	Peer address	Connection ID	Active stream count	Active streams	Total stream count	Packets Sent	Packets Lost	Packets Received	Connected
25.client-channel.google.com:443	QUIC_VERSION_46	74.125.195.189:443	8ebfcc9213edd3e	1	181	89	1601	0	1784	true
play.google.com:443	QUIC_VERSION_46	172.217.3.174:443	60c38f3a74d5fced	0	None	1	6	0	4	true
play.google.com:443	QUIC_VERSION_46	172.217.3.174:443	93474aaf52bf8a81	0	None	2	6	0	5	true
ssl.gstatic.com:443	QUIC_VERSION_46	172.217.3.195:443	8f847fe49a1bd5d8	0	None	1	5	1	4	true

Figure 43 QUIC Active Connections log


```

t=214765 [st= 0] QUIC_SESSION_PACKET_RECEIVED
--> peer_address = "74.125.195.189:443"
--> self_address = "192.168.0.12:61477"
--> size = 41
t=214765 [st= 0] QUIC_SESSION_UNAUTHENTICATED_PACKET_HEADER_RECEIVED
--> connection_id = "0"
--> header_format = "IETF_QUIC_SHORT_HEADER_PACKET"
--> packet_number = 1784
--> reset_flag = 0
--> version_flag = 0
t=214765 [st= 0] QUIC_SESSION_PACKET_AUTHENTICATED
t=214765 [st= 0] QUIC_SESSION_STREAM_FRAME_RECEIVED
--> fin = false
--> length = 22
--> offset = 176
--> stream_id = 181
t=214790 [st=25] QUIC_SESSION_ACK_FRAME_SENT
--> delta_time_largest_observed_us = 25659
--> largest_observed = 1784
--> missing_packets = []
--> received_packet_times = []
t=214791 [st=26] QUIC_SESSION_PACKET_SENT
--> encryption_level = "ENCRYPTION_FORWARD_SECURE"
--> packet_number = 1601
--> sent_time_us = 91025798925
--> size = 29
--> transmission_type = "NOT_RETRANSMISSION"

```

6. As you can see, right above is the screenshot capturing the whole QUIC process of establishing a session -starting from sending packets and acknowledgments to receiving packets, header protection, AEAD. The screenshot below is a form of a request to the server (like the content of an HTML form). It generally specifies the content size (261 in our case). It is a 'POST' request, which means the data is being pushed to the server (creating a source). To achieve in-order delivery, priority frames are sent on the control stream (quic_priority= 1 in our case). An endpoint usually keeps a limit on the cumulative number of incoming streams a peer that can be opened, in

order to control Concurrency. The limiting factor is :

(stream ID) < (max_stream *4 + initial_stream_id_for_type)

```
t=212997 [st= 0] QUIC_CHROMIUM_CLIENT_STREAM_SEND_REQUEST_HEADERS
--> :method: POST
    :authority: play.google.com
    :scheme: https
    :path: /log?format=json&hasfast=true
    content-length: 261
    content-encoding: gzip
    sec-fetch-dest: empty
    user-agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/80.0.3987.163 Safari/537.36
    content-type: application/binary
    accept: */*
    origin: https://ogs.google.com
    x-client-data: CjG2yQEIorbJAQjBtskBCKmdygEIy67
    sec-fetch-site: same-site
    sec-fetch-mode: cors
    referer: https://ogs.google.com/
    accept-encoding: gzip, deflate, br
    accept-language: en-US,en;q=0.9
--> quic_priority = 1
--> quic_stream_id = 7
```

7. The screenshot below depicts the certificate transparency- a standard for auditing and monitoring digital certificates. There are three ways that this can be done-
 1. Through an Embedded SCT (like in our case).
 2. Through an OCSP (Online Certificate Status Protocol) response.
 3. Through a TLS extension.

You can also see the encryption algorithm used here is the SHA-256 hash algorithm.

```

12023 [st= 0] SIGNED_CERTIFICATE_TIMESTAMPS_RECEIVED
--> embedded_scts = "APEAdgCyHgXMi6LNiiBOh2b5K7mKJSBna9r6c0eySVMt74uQXgAAAXELg/duAAAEAwBHMEUCIGjDyhYIqX+pEe4cl
--> scts_from_ocsp_response = ""
--> scts_from_tls_extension = ""
12023 [st= 0] SIGNED_CERTIFICATE_TIMESTAMPS_CHECKED
--> scts = [{"extensions":"","hash_algorithm":"SHA-256","log_id":"sh4FzIuizYogTodm+Su5iiUgZ2va+nDnsk1TLe+LkF4:
12024 [st= 1] CERT_CT_COMPLIANCE_CHECKED

```

The screenshot below depicts how the encrypted version of the certificate looks like when retrieved:

```

-----BEGIN CERTIFICATE-----
MIIE9DCCA9ygAwIBAgIRAJFSbYJ0ckG3CAAAAAA17VMwDQYJKoZIhvcNAQELBQAw
QjELMAkGA1UEBhMCVVMxHjAcBgNVBAoTFUdvd2dsZSBUcnVzdCBTZXJ2aWNlczET
MBEGA1UEAxMKR1RTIENBIDFPMTAeFw0yMDAzMjQwNDlaFw0yMDA2MjYwNjQ4
NDlaMGcxXzAjbGVBAYTA1VTRMRWEQYDVQQIEWpDYWxpZm9ybmhmRYWFAyDVQqH
Ew1Nb3VudGFpbWwV3MRMRWEQYDVQQKEWpHb29nbGUgTEExDMRYWFAyDVQqDDA0q
LmdzdGF0aWMuY29tMFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAE8TLFz1MHx5XZ
9hJOUMziJuCbbtnXODV4oG36D+K5BMEICGo1Nn2yqcY8lmZF2Srov8t1f7DFuSS7
YlAxHHBbpqOCAokwggKFMA4GA1UdDwEB/wQEAwIHgDATBgNVHSUEDDAKBggrBgEF
BQcDATAMBGNVHRMBAf8EAJAAMB0GA1UdDgQWBBCqHQ6vf0RuZk0WwiIE//oJZ9nq
OjAfBgNVHSMEGDAWgBSY0fhUE0vPm+xgnxiQG6DrfQn9KzBkBggrBgEFBQcBAQRY
MFYwJWYIKwYBBQUHMAAGGG2h0dHA6Ly9vY3NwLnBraS5nb29nL2d0czFvMTArBggr
BgEFBQcwAoYfaHR0cDovL3BraS5nb29nL2dzcjIvR1RTMU8xLmNydDBNBGNVHREE
RjBEgg0qLmdzdGF0aWMuY29tgggqLmtuLmRldoIUKi5tZXRYaWMuZ3N0YXRpYy5j
b22CC2dzdGF0aWMuY29tggZrbi5kZXIwIQYDVQR0gBBowGDAIBgZngQwBAGIwDAYK
KwYBBAHWeQIFAzAvBgNVHR8EKDAmMCSgIqAggh5odHRwOi8vY3JsLnBraS5nb29n
L0dUUzFPM5jcmwwggEFBgorBgEEAdZ5AgQCBIH2BIHzAPEAdgCyHgXMi6LNiiBO
h2b5K7mKJSBna9r6c0eySVMt74uQXgAAAXELg/duAAAEAwBHMEUCIGjDyhYIqX+p
Ee4cDQj6Tb/MCQXH5IX51/aRGIUN+tVmAiEAhxj+HdwAQEyH06SPzDnE10qPdWLx
6ZO+b8bb5Zh7fdsAdwBep3P531bA57U2SH3QSeAyepGaDISHhKEGHWWgXFFWAAA
AXELg/ejAAAEAwBIMEYCIQCbNjMLoXRjvjshS17uT6G/2ydrtttrikXA81+xVlhIa
/gIhAM6Xk59PE8fGXEEMXgKXfMjDzK4NzZkAgL00rGs3SHHZ/MA0GCSqGSIb3DQEB
CwUAA4IBAQC5WT7gNcVzSwQ94qT7M3iePegkNISSOLWE0BeF+QzEz47bg19JdDbB
qb+9uf0HXvBRJu9MMPnGhm1JBVPYBgdkrU7i/+GeE0pdhnf7p32Yi1Udi4v/dMU3
01Q+iFjsMVx/IQc+egCxiMtoRr/Rb3gkZfjgQThKDVEstATVsmqpWk5/1Lfhyq/C
jT01xJbbRPaMpI3VDI1qnnaVmQXewDJyvXF+9d0rbn45whgcCDg1dB35Br1/Utcz
BPzQHPCZNgKRVY0VR/p3KpCQzee+jPOKfXo9q3GVkj1b/+Ea+s34RMXVCC33xqhi
dBiLsWxKti3w5+Iis+hn29NIOeW2KE3R
-----END CERTIFICATE-----

```

direct:// (transport_socket_pool)								
Name	Pending	Top Priority	Active	Idle	Connect Jobs	Backup Timer	Stalled	
ssl/ca-central-1.console.aws.amazon.com:443	0	-	0	<u>1</u>	0	stopped	false	
ssl/console.aws.amazon.com:443	0	-	0	<u>1</u>	0	stopped	false	
ssl/ma.ttias.be:443	0	-	1	0	0	stopped	false	
ssl/phd.aws.amazon.com:443	0	-	0	<u>1</u>	0	stopped	false	
ssl/www.reddit.com:443	0	-	1	0	0	stopped	false	

Figure 44 Transport Socket Pool logs

Lastly, what you see above is the transport socket pool listing all the active websites (Active =1) and (Active=0) for the inactive ones. All the sites use UDP QUIC port number 443.

Using TLS FOR QUIC

QUIC takes responsibility for the integrity and confidentiality protection of packets. To accommodate this, it uses TLS handshake derived keys, but instead of carrying TLS records over QUIC (as with TCP), TLS Handshake and Alert messages are carried directly over the QUIC transport, which takes over the responsibilities of the TLS record layer.

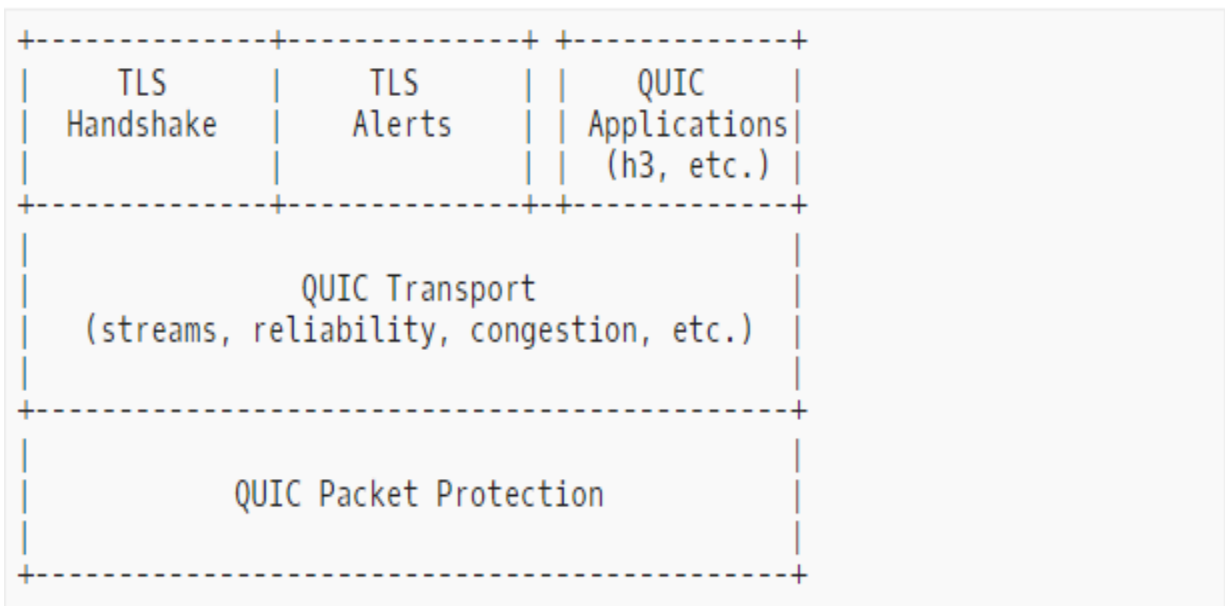


Figure 45 Using TLS for QUIC (IETF)

QUIC also relies on TLS for negotiation of parameters and authentication since they are critical to performance and security.

Rather than adopting a strict layering, the two protocols cooperate well:

- QUIC makes use of the TLS handshake;
- TLS makes use of the reliability, the record layer, and the ordered delivery by QUIC.

At a prominent level, there exist two main interactions between the QUIC and TLS components:

- With QUIC giving a reliable stream abstraction to TLS, The TLS component sends and receives messages via its component.
- The TLS component provides the QUIC component with a series of updates, such as (a) new packet protection keys to install (b) state changes such as the server certificate, handshake completion, etc.

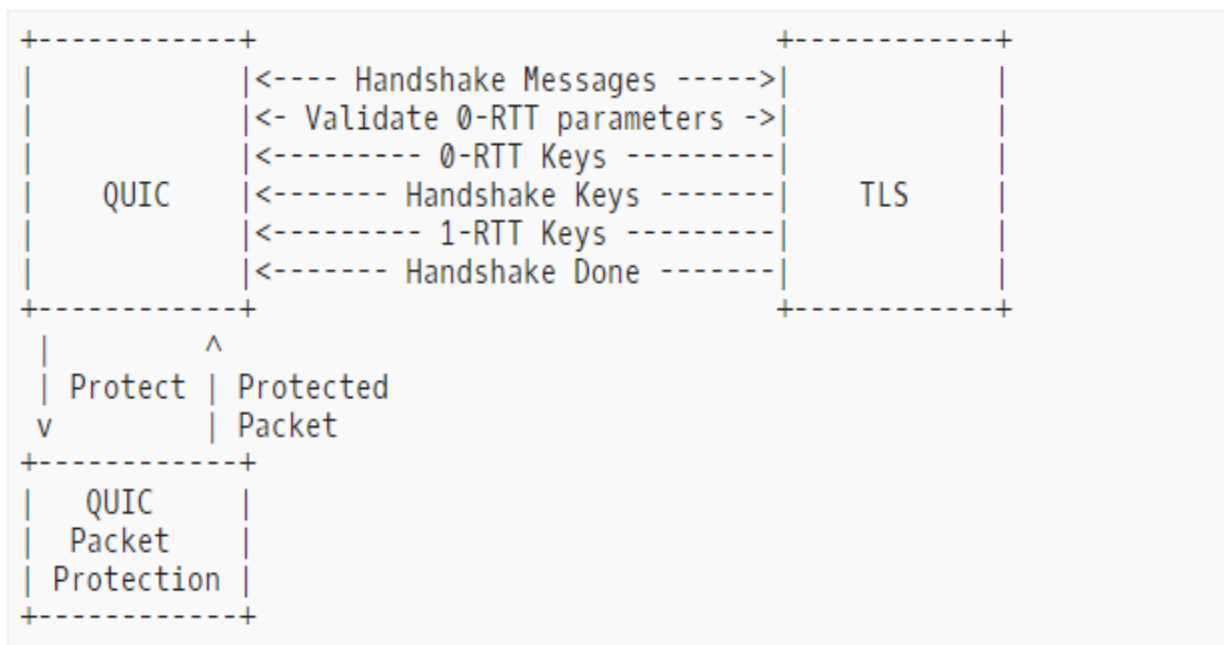


Figure 46 Interactions between QUIC and TLS (IETF)

The QUIC applications send data as QUIC STREAM frames rather than sending it through TLS "application_data" records (like in TLS over TCP). These are then carried in QUIC packets.

Thus, the interface from TLS to QUIC consists of four primary functions [15]:

- Exchange of handshake messages.
- Processing stored application and transport state from an already resumed session and determining if it is valid to accept early data.

- Rekeying (both transmitting and receiving)
- Updates of handshake states.

Handshaking process:

QUIC_SESSION_CRYPTO_HANDSHAKE_MESSAGE_SENT

--> CHLO<

```

SNI : "ssl.gstatic.com"
STK : 0x17eeddb3288d8a79bfff7253cb4c16735af2f38e791c9ae22f1eb7185c3b0bb31ffe6da2e736b2ab0685fb2e51bc
VER : 'Q046'
CCS : 0x01e8816092921ae87eed8086a2158291
NONC: 0x5e91802a3030303030303030ae12d2a91c20e8a16d4d8ac6f2d5b4701c1d61e1
AEAD: 'AESG'
UAID: "Chrome/80.0.3987.163 Windows NT 10.0; Win64; x64"
SCID: 0x6ab035275cd99b84eb2decea1c0baba2
TCID: 0
PDMD: 'X509'
SMHL: 0x01000000
ICSL: 30
NONP: 0x42d804ad57c7235650f68019306c16310c46bb376629b485f4de22c1ae8092c4
PUBS: 0x13222d2170ff7012902420a0ce70275ce4261a9b60ae93eda95c895b7501b167
MIDS: 100
SCLS: 1
KEXS: 'C255'
XLCT: 0x20c5a5d2ffa22dc0
CSCT: 0x
COPT:
CCRT: 0x20c5a5d2ffa22dc06032cb92a0414ddf
IRTT: 48368
CFCW: 15728640
SFCW: 6291456

```

The screenshot above shows how the initial Client Hello (CHLO) looks like during a network transmission. The packet contains a server configuration with it's Diffie Hellman public value, a certification chain authenticating the server etc. In order to process the handshake, TLS depends on being able to exchange handshake messages. Their basic functions on this interface are: one where QUIC provides handshake packets and one where QUIC requests handshake messages. QUIC provides TLS with the transport parameters that it wishes to carry right before starting the handshake.

At first, a QUIC client requests handshake bytes from TLS. Therefore, the client gets the required handshake bytes right before sending its first packet. A QUIC server begins with the process by providing the client's handshake bytes to TLS. The TLS stack at an endpoint would have a current sending and receiving encryptions, respectively, at any time. Each encryption level that is reliably transmitted to the peer in CRYPTO frames is associated with a different flow of bytes. When handshake bytes to be sent are provided by the TLS, any packet that includes the CRYPTO frame is protected using keys from the corresponding encryption level, and the handshake bytes are appended to the current flow. The unprotected content of TLS handshake records is taken by QUIC as the content of CRYPTO frames. The CRYPTO frames are assembled into QUIC packets, and these packets are protected using QUIC packet protection. TLS record protection is not specifically used by QUIC.

QUIC's capability of conveying TLS handshake records is limited to CRYPTO frames.

When a QUIC packet containing a CRYPTO frame is received by the endpoint from the network, it proceeds as follows [15]:

- Sequence the data into an input flow if the packet was a part of the TLS receiving encryption level. The offset is utilized to find the proper location in the data sequence with all stream frames. If this process results in discovering the availability of new data, then that data is delivered to TLS in order.
- The packet from a previously installed encryption level should not contain data that extends past the end of the previously received data in that flow. Implementations should treat any violations of this sort as a connection error of type `PROTOCOL_VIOLATION`.
- If the packet is a part of a new encryption level, it is then saved for later processing by TLS. The saved data can be provided only if the TLS moves to 'receive' from this encryption level. While providing data from any new encryption level to TLS, it is important to check whether there is data from a previous encryption level that TLS has not consumed. If so, this should be treated as a connection error of type `PROTOCOL_VIOLATION`.

The TLS and QUIC are continuously communicating. When TLS is provided with new data, new handshake bytes are requested from TLS immediately after. TLS may not be able to provide any bytes if the received handshake messages are incomplete, or it simply has no data to send. Once the TLS handshake is completed, QUIC is informed about it, along with any remaining final handshake

bytes that TLS may need to send. The Transport parameters advertised by the peer during the handshake are also provided to QUIC by TLS[15].

Encryption Levels:

TLS provides QUIC with the keys for the new encryption levels as they become available. Furthermore, TLS indicates to QUIC that reading or writing keys at that encryption level is available when keys at a given encryption level become available to TLS. These events always occur right after the TLS is provided with new handshake bytes, or after TLS itself produces handshake bytes. Hence, these events are not asynchronous. ¶

As a new encryption level becomes available, TLS provides QUIC with three items [15]:

- A secret function.
- An Authenticated Encryption with Associated Data (AEAD) function for protection.
- A Key Derivation Function (KDF)


```

t=212022 [st=0] +QUIC_STREAM_FACTORY_JOB [dt=2]
                  --> server_id = "https://ssl.gstatic.com:443"
t=212022 [st=0]   QUIC_STREAM_FACTORY_JOB_BOUND_TO_HTTP_STREAM_JOB
                  --> source_dependency = 34394 (HTTP_STREAM_JOB)
t=212022 [st=0] +HOST_RESOLVER_IMPL_REQUEST [dt=1]
                  --> allow_cached_response = true
                  --> dns_query_type = 0
                  --> host = "ssl.gstatic.com:443"
                  --> is_speculative = false
                  --> network_isolation_key = "null"
t=212022 [st=0]   HOST_RESOLVER_IMPL_IPV6_REACHABILITY_CHECK
                  --> cached = true
                  --> ipv6_available = false
t=212022 [st=0]   HOST_RESOLVER_IMPL_CREATE_JOB
t=212022 [st=0]   HOST_RESOLVER_IMPL_JOB_ATTACH
                  --> source_dependency = 34396 (HOST_RESOLVER_IMPL_JOB)
t=212023 [st=1] -HOST_RESOLVER_IMPL_REQUEST
t=212023 [st=1]   QUIC_STREAM_FACTORY_JOB_CONNECT [dt=1]
                  --> require_confirmation = false
t=212024 [st=2] -QUIC_STREAM_FACTORY_JOB

```

http_stream_job corresponds to httpStreamFactory Job- one request can have multiple jobs. It includes the DNS and proxy lookups. HTTP_STREAM_JOB log events are separate from URL_REQUEST because two-stream jobs can be created at a time and races against each other, in our case- one for QUIC and one from TCP.

A Gist of the Entire Connection establishment process (0-RTT and 1-RTT)

1. ClientHello

The client's first Initial packet contains the start of the first cryptographic handshake message, which is ClientHello for TLS. QUIC packet and the associated framing add at least 36 bytes of overhead to the ClientHello message. If a connection ID without zero length is chosen by the client, that overhead increases [15].

2. Peer authentication

TLS permits the server to request client authentication and provides server authentication. The identity of the server must be authenticated by the client. This requires that the certificate is issued by a trusted entity and that the verification involving the identity of the server is a part of the certificate [15].

3. 0-RTT

Servers usually send a NewSessionTicket message containing the "early_data" extension with a max_early_data_size of 0xffffffff to communicate their willingness to process 0-RTT data. There is a set limit on the amount of data that can be sent by the client in 0-RTT. This is controlled by the "initial_max_data" transport parameter supplied by the server. By sending an 'early_data extension' in the EncryptedExtensions, the server accepts the 0-RTT. The server then acknowledges and processes the 0-RTT packets received by it. A 0-RTT is rejected by the server by sending the EncryptedExtensions without an early_data extension. If the 0-RTT is rejected, all connection characteristics such as transport parameters, choice of application protocol, and any application configuration assumed by the client may be incorrect. Therefore, the client should reset the state of all its streams, including application states that are bound to those streams [15].

4. Errors

When you convert the one-byte alert description into a QUIC error code, a TLS alert is turned into a QUIC connection. The alert description is then made a part of the 0x100 to produce a QUIC error code. This is done from the range reserved for CRYPTO_ERROR. The resulting value is sent in as a QUIC CONNECTION_CLOSE frame[15].

Packet Protection and Design Protection:

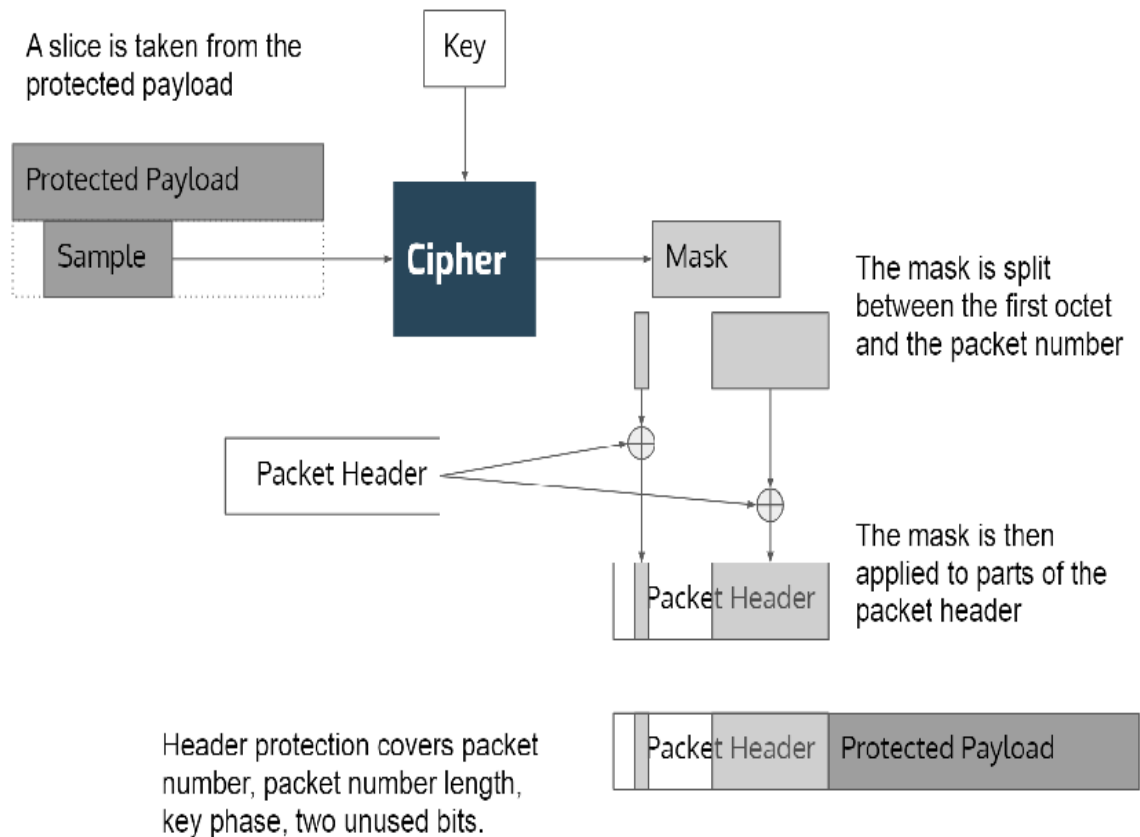


Figure 47 Packet protection in QUIC

Packet protection keys in QUIC:

According to the latest version of the IETF documentation on QUIC, QUIC is said to derive packet protection keys similar to how the TLS derives record protection keys. Each encryption level is said to have separate secret values for the protection of packets sent in each direction. These traffic secrets which are used by QUIC for all encryption levels except the Initial encryption level is derived by TLS. The secrets required for the **Initial encryption level** are calculated entirely based on the client's initial Destination Connection ID.

Below is the screenshot of how the initial encryption level is logged when the network traffic is analyzed. This is part of the QUIC session packet that is sent.

```

12024 [st= 1]    QUIC_SESSION_PACKET_SENT
--> encryption_level = "ENCRYPTION_INITIAL"
--> packet_number = 1
--> sent_time_us = 91023032487
--> size = 1350
--> transmission_type = "NOT_RETRANSMISSION"

```

The **packet protection keys** are computed using the KDF provided by TLS from the TLS secrets. The HKDF Expand Label Function is used to calculate client in, server in, quic key, quic iv, and quic hp values. This function is ideally used to expand the generated output of a random input (such as an existing shared key) into a much larger cryptographically independent output, thereby producing multiple keys from that initial shared key, such that the same process may reproduce the same secret keys really safely on several devices, as long as the same inputs are used [15].

The label “quic key” and the current encryption level secret are extremely important inputs to the KDF to produce the AEAD key; the label "quic iv" is used to derive the IV. The header protection key makes use of the "quic hp" label. The use of these labels provides key separation between QUIC and TLS.

The KDF used for initial secrets in the case of TLS Architecture is always the HKDF-Expand-Label function from TLS 1.3. The keys produced as a result of this function look something like the following:

The example labels generated by the HKDF-Expand-Label function are [15]:

client in: 00200f746c73313320636c69656e7420696e00

server in: 00200f746c7331332073657276657220696e00

quic key: 00100e746c7331332071756963206b657900

quic iv: 000c0d746c733133207175696320697600

quic hp: 00100d746c733133207175696320687000

The initial secret is common:

```

initial_secret = HKDF-Extract(initial_salt, cid)
= 524e374c6da8cf8b496f4bcb696783507aaf6e6198b202b4bc823ebf7514a423

```

The secrets for protecting client packets are:

`client_initial_secret`

`= HKDF-Expand-Label(initial_secret, "client in", _, 32)`

`= fda3953aecc040e48b34e27ef87de3a6`

`098ecf0e38b7e032c5c57bcbd5975b84`

key = `HKDF-Expand-Label(client_initial_secret, "quic key", _, 16)`

`= af7fd7efebd21878ff66811248983694`

iv = `HKDF-Expand-Label(client_initial_secret, "quic iv", _, 12)`

`= 8681359410a70bb9c92f0420`

hp = `HKDF-Expand-Label(client_initial_secret, "quic hp", _, 16)`

`= a980b8b4fb7d9fbc13e814c23164253d`

The secrets for protecting server packets are:

`server_initial_secret`

`= HKDF-Expand-Label(initial_secret, "server in", _, 32)`

`= 554366b81912ff90be41f17e80222130`

`90ab17d8149179bcadf222f29ff2ddd5`

key = `HKDF-Expand-Label(server_initial_secret, "quic key", _, 16)`

`= 5d51da9ee897a21b2659ccc7e5bfa577`

iv = `HKDF-Expand-Label(server_initial_secret, "quic iv", _, 12)`

`= 5e5ae651fd1e8495af13508b`

hp = HKDF-Expand-Label(server_initial_secret, "quic hp", _, 16)
= a8ed82e6664f865aedef6106943f95fb8

AEAD (Authentication Encryption with Associated Data) Protection:

Packets are generally protected prior to applying header protection. The packet header that is unprotected is part of the associated data (A), which is sent. When removing packet protection, the receiver endpoint first removes the header protection. Prior to establishing a shared secret between the two communicating parties, packets are protected with AEAD_AES_128_GCM and a key derived from the Destination Connection ID in the client's first Initial packet. All QUIC packets other than the Version Negotiation and Retry packets are protected with an AEAD algorithm, which is actually a drawback offered by this protocol.

The IV and the key for the packet are calculated using the current encryption-level secret and the label, "quic key."

Here, the nonce, N= Packet protection IV + packet number

When the QUIC packet number is reconstructed, the 62 bits of that packet number in a network byte order are all left-padded with zeros to the size of the packet protection IV.

AEAD nonce= padded packet number (Exclusive OR) Packet Protection IV

.The 'A' (Associated data) for the AEAD is nothing but the contents of the QUIC header, starting from the flags byte in either the long or short header, including and up to the unprotected packet number. The 'P'(input plaintext) for the AEAD is the payload of the QUIC packet. The output 'C' (Ciphertext) of the AEAD is transmitted in place of P [15].

Header Protection:

Certain parts of QUIC packet headers, particularly the Packet Number field, are protected using a key that is derived separately to the IV and the packet protection key. Protection of confidentiality for those fields that are not completely exposed to on-path elements is provided by the keys derived using the "quic.hp" label.

This protection ideally applies to the least-significant bits of the first byte, plus the Packet Number field. The four least-significant bits of the very first byte are protected for packets with **long headers**; on the other hand, the five least significant bits (LSB) of the first byte are protected for

packets with **short headers**. For both the types of header forms used, this covers the reserved bits and the Packet Number Length field; and For packets with a short header, the Key Phase bit is also completely protected.

In order for the header protection to be used to protect the key phase, the same header protection key is used for the entire duration of the connection with the value unaffected even after a key update.

Referring to the last research paper, we discussed in the previous section (), the proposed packet tracking architecture consists of multiple passive measurement probes deployed in the network. **The proposed idea here is to deploy one such measurement probe (like impd4e) within the header protection of a sent packet.** In this way, each and every sent packet can be tracked, overcoming the disadvantage of their proposed architecture- Only a certain number of packets (selected by a hash algorithm) can be tracked.

Header protection is followed by packet protection. The encryption algorithm uses the sampled ciphertext of the packet as the input. The encryption algorithm selected depends upon the negotiated AEAD.

The output of this algorithm is a 5-byte mask which undergoes an exclusive OR operation with the protected header fields. The least significant bits (LSB) of the first mask byte masks the least significant bits of the first byte of the packet, and the remaining bytes mask the packet number. ¶

Given below is a sample algorithm from the latest IETF document outlining the concepts of QUIC and TLS [15] for applying header protection (long and short headers). Removing the header protection differs in the order in which the packet number length (pn_length) is determined.

```
mask = header_protection(hp_key, sample)
```

```
pn_length = (packet[0] & 0x03) + 1
```

```
if (packet[0] & 0x80) == 0x80:
```

```
    # Long header: 4 bits masked
```

```
    packet[0] ^= mask[0] & 0x0f
```

```
else:
```

```
# Short header: 5 bits masked

packet[0] ^= mask[0] & 0x1f

# pn_offset is the start of the Packet Number field.

packet[pn_offset:pn_offset+pn_length] ^= mask[1:1+pn_length]
```

Below is the screenshot of the analysis of the QUIC protocol in operation. The QUIC session logged all the events that contribute to the transmission of the QUIC packets. The two snapshots below describe how the short and long header packets are recorded during an actual network transmission. The long header packet is 0-RTT protected, and the short header packet contains the application data.

Short Header:

```
+++++
|0|1|S|E E E E|
+++++
|               Destination Connection ID (0/32..144)           ...
+++++
```

```
13038 [st=41] QUIC_SESSION_UNAUTHENTICATED_PACKET_HEADER_RECEIVED
--> connection_id = "0"
--> header_format = "IETF_QUIC_SHORT_HEADER_PACKET"
--> packet_number = 5
--> reset_flag = 0
--> version_flag = 0
```

Long Header:

```
+++++
|1|1|T T|E E E E|
+++++
|               Version -> Length Fields                       ...
+++++
```



```

12068 [st= 45]    QUIC_SESSION_UNAUTHENTICATED_PACKET_HEADER_RECEIVED
                  --> connection_id = "0"
                  --> header_format = "IETF_QUIC_LONG_HEADER_PACKET"
                  --> long_header_type = "ZERO_RTT_PROTECTED"
                  --> packet_number = 1
                  --> reset_flag = 0
                  --> version_flag = 1
-                -

```

Now let's jump into the proposed idea. The idea here is to use the **measurement probe** along with the `hp_key`:

```
mask = header_protection(hp_key, sample, probe_function)
```

When removing the packet protection, an endpoint first removes this element and the header protection. This way, the client will know if the packet has reached the destination without being tampered with or not.

Let us now consider using a specific AEAD algorithm, such as AEAD_CHACHA20_POLY1305. When this is in use, header protection uses the raw ChaCha20 function. This algorithm makes use of 16 bytes and a 256-bit key sampled from the packet protection output. The sampled ciphertext contains the block counter in its first 4 bytes. The byte sequence is portrayed as a little-endian value only in one case- when a ChaCha20 implementation takes a 32-bit integer in place of a byte sequence.

The remaining 12 bytes of the packet protection output is used as the nonce. The nonce bytes are portrayed as a sequence of 32-bit little-endian integers only in one case- when a ChaCha20 implementation might take an array of three 32-bit integers in place of a byte sequence.

The encryption mask is composed by imploring ChaCha20 to protect 5 zero bytes. Involving the measurement probe in the pseudocode:

```
counter = sample[0..3]
```

```
nonce = sample[4..15]
```

```
mask = ChaCha20(hp_key, probe_function, counter, nonce, {0,0,0,0,0})
```

Header protection Analysis of the packets:

NAN() analyzes the authenticated encryption algorithms that provide the required nonce privacy, referred to as "Hide Nonce" (HN) transforms. The general-header protection construction discussed in this report is one of the Hide Nonce transforms algorithms (HN1). Header protection utilizes the output of the packet protection AEAD (Authenticated Encryption with Associated Data) to produce a sample, and then encrypts the entire header field with the help of a pseudorandom function (PRF):

$$\text{protected_field} = \text{field XOR PRF}(\text{hp_key}, \text{sample})$$

The header protection types considered here use a pseudorandom permutation (PRP) in place of a generic PRF. However, since all PRPs are also PRFs, these variants are not much different from the HN1 construction. As hp_key is completely different from the packet protection key, it follows that header protection, therefore, guarantees the privacy of the field and the protected packet header. To ensure equivalent security guarantees, future header protection variants based on this construction **MUST** use a PRF.

Usage of the same key and ciphertext sample more than once risks compromising the entire header protection leading to security issues. In order to get the exclusive OR value of the protected fields, it is suggested to protect two different headers with the same ciphertext and the same key.

To prevent the packet headers from being modified by an attacker, authentication using packet protection is applied to the header transitorily. This means that the entire packet header is included in the authenticated additional data too. Only once the packet protection is removed, the protection fields that are modified or falsified are detected.

The packet measurement probe to be used will be similar to that of impd4e used in the last discussed research paper (). It will consist of a small network probe that will allow us to monitor the network path. The code for the imd4e is open source and uses simple C programming. It can be therefore tailored to our need to incorporate it with the header protection like a tracker. For ease, the code can also be modified using python programming. Using the IPFIX protocol, it can export packet IDs and relevant information to the sender (the sender will host a library for exporting measurement results via the IPFIX standard).

Challenges:

1. **Increased payload-** One of the main downsides of this proposal will be increased payload due to the measurement probe function and a library to extract the location information.
2. **Problems in specificity-** Each network is unique in terms of the number of users, devices, traffic, usage, etc. Therefore, it will be a challenge to come up with a customizable code that caters to tracking in networks big and small.
3. **Increased Complexity-** Since we are suggesting an addition to the already existing architecture, complexity is bound to increase. This is one of the reasons why this proposal is limited to the QUIC protocol for the time being. It being a fairly new protocol, is open to new research.
4. **Ambiguity and scattered documentation-** QUIC is an experimental protocol that is currently being used only in very few applications, mainly due to the underlying security issues and ambiguity because of a number of scattered documentations.
5. **QUIC is very fast-evolving-** With the advent of technology and plenty of resources out there, every day is a step towards bug fixes, improvements in the architecture, etc. Google held its very first QUIC working group meeting in the year 2016. These 4 years since then have seen the biggest leap in its usage. Developing a complex system over such a fast-evolving protocol will not be easy at all.

Why TyT then?

Given the above challenges, one can get skeptical about using this concept, but again every technology comes with a trade-off. Security is one of the major issues we are facing today. With TyT, issues related to Security and Latency problems are getting a common solution. Initially, this proposal has been limited to the QUIC protocol only because it is a fairly new protocol, and hence, it can be experimented with before standardizing it.

According to Mattias Geniar, if QUIC features prove effective, those features should migrate into a later version of TCP. Imagine TCP with no three-way handshakes and no requirement of acknowledgments. This could mean faster processing speeds and transmission rates. I reiterate, the idea is to provide more control to the sender, to understand how the sent packet traverses, detect attacks like man in the middle, DDoS, and get a clearer picture of what happens to your sent packet.

Since the advent of Software Defined Networking, all the major processes are transforming into a software version. A programmed measurement probe, such as, the `imped4e` is a Light OpenSource network measurement probe that uses `pcap` to generate `packetID` for traversing packets for packet tracking and hop-by-hop delay measurements. Hash-based packet selection is also supported, and results are exported via `IPFIX`. The idea here is not to develop a map (like google maps) but rather learn as you traverse through the network. This `imped4e` is available openly on the GitHub, and the source code is in an easy to understand C language. By using python, we could make this code even simpler and cater to our needs of inserting it with the header protection in the transport layer. It's never a far-fetched idea when the security of a network is at stake.

Conclusion:

Cloud Computing is entirely changing the way businesses today consume computer services. Approximately 75% of the businesses today operate over the cloud. Everything is available from anywhere at the snap of the finger. Well, the major stranglehold for this vast technology is a security threat. Cloud is like a reservoir of data today, which means illegal access to this reservoir can lead to data breaches and intrusion of privacy. Hence, securing this data takes is of prime importance.

In this project, we first discussed what cloud computing is, the security threats it faces in today's world, and then we went on to discuss a few attacks and the corresponding mitigation techniques in place. We then went on to ponder over some research ideas in the areas of cloud security, QUIC protocol, and it's security. One of the newest discussions was the idea of using QUIC in cloud-QUIC.cloud. It is still in the nascent stage and has a lot of evolving to do. It has the potential to be a stringent competitor in the market of cloud providers owing to its simplicity and transmission speeds. We also go on to understand the working, architecture, and security considerations of the QUIC protocol with an analysis of the network traffic. According to Google, on mobile Android devices, it is claimed that QUIC has helped to reduce the latency of Google Search responses by 3.6% and YouTube video buffering by 15.3%. With this track record and many more security glitches being fixed over time, we can only expect the rise of this particular protocol. In fact, it may become a good choice over TCP, although this might take several decades. Finally, before jumping into the main idea, we try to understand how the TLS Security Architecture is incorporated in the QUIC protocol.

Solving the issue of Security at a protocol level is what this capstone project goes for. The key idea is to improve efficiency, security (combat attacks like Man in the Middle, DDoS), improve latency, and to provide more control to the sender of a packet. By extending and using the concept of Multi-Hop Packet tracking in experimental facilities in the TLS architecture used to secure protocols like QUIC, we can achieve all of the above. Today, with the advent of Software Defined Networking, coding a measurement probe to track packets does seem like a not-so-far reality.

REFERENCES:

1. Sara Bouchenak, Gregory Chockler, Hana Chockler, Gabriela Gheorghe, Nuno Santos, Alexander Shraer- **Verifying Cloud Services- Present and the Future**
2. The cloud is here: embrace the transition - How organizations can stop worrying and learn to “think cloud”- Deloitte.
3. Adam Langley, Alistair Riddoch, Alyssa Wilk, Antonio Vicente, Charles Krasic, Dan Zhang, Fan Yang, Fedor Kouranov, Ian Swett, Janardhan Iyengar, Jeff Bailey, Jeremy Dorfman, Jim Roskind, Joanna Kulik, Patrik Westin, Raman Tenneti, Robbie Shade, Ryan Hamilton, Victor Vasiliev, Wan-Teh Chang, Zhongyi Shi- **Google- The QUIC Transport Protocol- Design and Internet-Scale Deployment.**
4. Robert Lychev, Samuel Jero, Alexandra Boldyreva, Cristina Nita-Rotaro- **How Secure and quick is QUIC?-Provable Security and Performance Analyses.**
5. Mohammadkazem Taram, Ashish Venkat, Dean Tullsen- Packet Chasing- **Spying on Network Packets using a Cache Side-channel**
6. Tacio Santos, Christian Henke, Carsten Schmoll, Tanja Zseby- **Multi-Hop Packet Tracking for Experimental Facilities.**
7. V. Sekar, and P. Maniatis. **Verifiable resource accounting for cloud computing services.** In Proceedings of the 3rd ACM workshop on Cloud computing security workshop, CCSW '11, pages 21{26, New York, NY, USA, 2011. ACM.
8. F. Zhou, M. Goel, P. Desnoyers, and R. Sundaram. Scheduler vulnerabilities and attacks in cloud computing. IEEE International Symposium on Networking Computing and Applications, 2011.
9. <https://github.com/tubav/impd4e>- impd4e Repository

10. Cloudflare blog- Road to QUIC- <https://blog.cloudflare.com/the-road-to-quic/>
11. **Google's QUIC protocol- moving the web from TCP to UDP-** <https://ma.ttias.be/googles-quic-protocol-moving-web-tcp-udp/>
12. QUIC- <https://en.wikipedia.org/wiki/QUIC>
13. QUIC.cloud- <https://www.quic.cloud/>
14. Quora-<https://www.quora.com/What-is-the-difference-between-IaaS-SaaS-and-PaaS?q=what%20is%20IaaS>
15. IETF document on using TLS with QUIC- <https://quicwg.org/base-drafts/draft-ietf-quic-tls.html#name-header-protection-sample>
16. Cloud Security Critical Analysis- https://www.researchgate.net/publication/331984452_Cloud_Security_Critical_analysis
17. Amazon Web Services
18. M. Fischlin and F. Gunther. 2014. Multi-Stage Key Exchange and the Case of Google's QUIC Protocol. In ACM Conference on Computer and Communications Security (CCS).
19. B. Ford. 2007. Structured Streams: A New Transport Abstraction. In ACM SIGCOMM.
20. J. Roskind. 2012. QUIC: Design Document and Specification Rationale. (2012)- <https://goo.gl/eCYF1a>.
21. M. van Dijk, A. Juels, A. Oprea, R. Rivest, E. Stefanov, and N. Triandopoulos. Hourglass schemes: how to prove that cloud _les are encrypted. In Proceedings of the 2012 ACM conference on Computer and communications security, pages 265{280. ACM, 2012.
22. Chromium QUIC Implementation. <https://cs.chromium.org/chromium/src/net/quic/>.

23. B. Claise. Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of IP Traffic Flow Information. RFC 5101 (Proposed Standard), January 2008.
24. J. Brutlag. 2009. Speed Matters. <https://research.googleblog.com/2009/06/speed-matters.html>.
25. V. Jacobson, R. Braden, and D. Borman. 1992. RFC 1323: TCP extensions for high performance. Internet Engineering Task Force (IETF) (1992).
26. Cyber Security Hub- <https://www.cshub.com/attacks/articles/top-5-cyber-security-breaches-of-2019-so-far>.
27. I.Indu, P.M Rubesh Anand, Vidhyacharan Bhaskar- **Identity and Access Management in a cloud environment- Mechanism and Challenges.**
28. Security Issues in Cloud Computing- McAfee- <https://www.mcafee.com/enterprise/en-ca/security-awareness/cloud/security-issues-in-cloud-computing.html>.
29. AWS Resource Groups.
30. QUIC Privacy and Security Workshop- <https://www.ndss-symposium.org/ndss2020/quic-privacy-and-security-workshop/>.
31. A security model and verified implementation of the QUIC Record Layer- https://github.com/chris-wood/NDSS20-QUIPS/blob/master/slides/quips2020_verified_implementation.pdf
32. QUIC Security- Martin Thomson- https://github.com/chris-wood/NDSS20-QUIPS/blob/master/slides/quips2020_security_overview.pdf.
33. QUIC at 10,000 feet- Google Developers.
34. Teaching good protocols to do bad things- <https://www.blackhat.com/docs/us-16/materials/us-16-Pearce-HTTP2-&-QUIC-Teaching-Good-Protocols-To-Do-Bad-Things.pdf>
35. Hkdf function- python- <https://github.com/casebeer/python-hkdf>.
36. Kevin D Broswe, Ari Juels, Alina Oprea- Proofs of Retrievability: Theory and Implementation
37. Intrusion Prevention Fundamentals- Signatures and Actions- <https://searchsecurity.techtarget.com/feature/Intrusion-Prevention-Fundamentals-Signatures-and-Actions>
38. Web Shells- China Chopper- <https://cyber.gc.ca/en/guidance/web-shells-china-chopper>.
39. ATT&CK Series- Credential Access- <https://www.optiv.com/blog/attck-series-credential-access>
40. Rapid Elasticity and the Cloud- <https://www.ibm.com/blogs/cloud-computing/2012/09/12/rapid->

[elasticity-and-the-cloud/](#).

41. SaaS, PaaS, and IaaS- a security checklist for cloud models-
<https://www.networkworld.com/article/2199393/saas--paas--and-iaas--a-security-checklist-for-cloud-models.html>.
42. What is different about Cloud Security?- <https://www.redhat.com/en/topics/security/cloud-security>.
43. Why the meteoric rise of Google QUIC is worrying mobile operators?-
<https://owmobility.com/meteoric-rise-google-quic-worrying-mobile-operators/>
44. Best practices to overcome Cloud Security Challenges- <https://www.cloudcodes.com/blog/cloud-security-challenges.html>.
45. Cloud Attacks- <https://attack.mitre.org/techniques/T1135/>.
46. Stefan Kruger, Sarah Nadi, Michael Reifz, Karim Ali, Mira Mezini, Eric Bodden, Florian Gopfert, Felix Gunther, Christian Weinert, Daniel Demmler, Ram Kamath- CogniCrypt- Supporting developers in using Cryptography.
47. Breach Incidence Response: An Emergency Preparedness Guide- DLA PIPER.