SWAZM

Short Technical Overview

Contributors: Vali Malinoiu

Table of contents

1. Abstract

The SWAZM.COM software introduces a new blockchain architecture designed to enable horizontal scaling of decentralized applications. This is achieved by creating a new blockchain with a reliable transfer network, storage capabilities, and compute containers. The software provides certificate authorities, authentication, storage and the scheduling of applications across many of CPU cores or clusters. The resulting technology is a blockchain architecture that may ultimately scale to sustain millions of applications, enable to write any kind of technology, and allows for quick and easy deployment and maintenance of decentralized applications.

2. Background

The first work on a cryptographically secured chain of blocks was described in 1991 by Stuart Haber and W. Scott Stornetta. They wanted to implement a system where documents timestamps could not be tampered with or backdated. In 1992, Bayer, Haber and Stornetta incorporated Merkle trees to the design, which improved its efficiency by allowing several documents to be collected into one block.

The first blockchain was conceptualized by a person (or group of people) known as Satoshi Nakamoto in 2008. It was implemented the following year by Nakamoto as a core component of the cryptocurrency Bitcoin, where it serves as the public ledger for all transactions on the network. Through the use of a blockchain, Bitcoin became the first digital currency to solve the double-spending problem without requiring a trusted authority and has been the inspiration for many additional applications.

At its simplest, the blockchain is a digital record stored on a network of computers around the world. Instead of securing information by restricting access, the blockchain shares information amongst all users. Ownership of funds (for example) is cryptographically verified, and the full transparency and mutual ownership of the system means that a bad actor is immediately recognizable as such and any transactions submitted by such a node are ignored.

The decentralized structure of the blockchain brings several key features in contrast to traditional centralized approaches:

- **Transparency**: it is possible for anyone to track the movement of funds from one account to another.
- **Immutability**: once confirmed, a transaction cannot be reversed. No one can interfere with a completed transfer.
- **Low cost:** transaction fees are minimal.
- **Cross-border:** funds can be sent as easily to someone on the other side of the world as they can to someone in the next room.

- **Speed:** due to the flat and transparent nature of the blockchain, transfers show up almost instantly and are typically confirmed in minutes, rather than hours or days.

3. DApps Constraints and Requirements

3.1 Support Millions of Users

To scale up and into mainstream, each Dapp must perform flawlessly, serving tens, if not hundreds of millions of users at the same time. There are use cases in which an application might not reach its potential or even work if a critical mass of users is not reached. A platform that can handle huge amounts of traffic is critical in order to compete with the existing business in the tech space.

3.2 Continuous Delivery

Businesses or developers need the flexibility to enhance their applications with new features or to deploy fixes to patch bugs or security holes. The platform needs to be robust enough to allow developers to easily push fast updates in order to solve as soon as possible any problem that may occur.

3.3 Fast Network Speed

Traditional applications today rely on a very short feedback loop, no more than a few seconds. Longer page loads, long responses and delays frustrate the end user. Fast network speeds are a huge advantage for the traditional application market. That's why it's important that the platform has an infrastructure which allows fast data transfers, no matter the latency or the distance between nodes.

3.4 Microservice Architecture

Big scale applications are in need of a microservice support to divide the workload across multiple CPUs or cluster of CPUs.

When it comes to “the right tool for the right job”, applications should not be limited to a certain programming language that isn’t robust enough to handle all the requirements. Instead, the right platform should allow the use of existing technologies that run on industry standard stacks and operating systems to aid in scaling the applications.

Contemporary applications use a multi-service multi-application architecture to work at a certain scale. In centralised environments, emerging technologies like Docker and ContainerD showcase that there is a clear need for scalable applications.

That's why the platform should have a built-in mechanism to orchestrate the microservice distribution and scaling. This will allow developers to fine-tune the execution and the location/s where the application is running.

3.5 Permission Authority

Most of the time, big scale applications come with a relatively big team attached, including ephemeral contractors. In this scenario, an authority is needed to add or revoke certificates for access to funds management, development, deployment or debugging an application.

This kind of permission management is critical to compete with the current traditional market. A platform that encapsulates this certificate management is pivotal in order to make the transition to a decentralised environment.

3.6 Localized Deployment

Deploying an application in the place where your target market resides is critical in creating the best user experience. The competition for user attention is at an all-time high and tech products that load faster, with smoother transitions and the lowest waiting times are prefered by the end user. In order to compete with centralised robust infrastructures, in a decentralised environment is critical to make use of geographical context when deploying an application.

3.7 Application Addressing and Load Balancing

An application needs:

- to be addressed in order to be found and used by its users (DNS addressing)
- to distribute the load inside the running microservices on the network (load balancing)

3.8 Decentralized Storage

Large storage providers have become dominant in the tech space by acting as trustworthy parties to store and transfer data on behalf of their clients.

This reality suffers from all the weakness associated with the trust-based model. The traditional cloud is vulnerable to a variety of security threats, like man-in-the-middle attacks, malware and infrastructure flaws which may expose sensitive data from private and corporate parties.

Compared to the above, a platform that offers a decentralized storage solution holds many advantages:

- data integrity, which can be maintained via Proof of Retrievability
- the impact of infrastructure failures and security breaches is greatly reduced
- data on the network is resistant to tampering, data failures and unauthorized access

3.9 Blockchain as Database

Running fully decentralized applications is challenging, given the current state of the technology. The infrastructure needed to make this work has to provide a hybrid database solution to marry the benefits of the blockchain (immutability, transparency and decentralisation) with the versatility of a traditional database.

On top of this, it needs to address multiple aspects:
-low latency, high capacity and high transaction rates
-withstand arbitrary failures
-it has to be Byzantine fault tolerant
-it has to be ACID compliant without having a single point of failure
-it needs to have multiple logical databases, so in case of a malicious attack where a party gains full access (admin level control), it could not delete all the data in the system with a single command
-traffic load balancing and data load balancing are needed so that big pieces of the same application's data don’t sit idle on a pool of nodes while other applications are generating huge traffic, essentially being a bottleneck for the whole network.

3.10 Open Market

Third-party providers for storage, compute and content delivery are controlling the market. Beyond the biggest tech players, more and more companies are relying on such providers to bring their product to the market. In this scenario, the rest of the cloud players are left drifting because individually they couldn't ever compete with the likes of AWS, Azure or Google Cloud.

With the rise of blockchain technology, we are entering an era where a platform that will level the playfield is needed. This will enable more players to compete in the same arena and for the same services using their existing hardware or devices. In turn, this may both drive down the cost and provide access to a wider array of services in a decentralised setting.

SWAZM

4. Overview

SWAZM is a powerful and flexible blockchain computing and storage engine which aims to use any existing unused computing power, storage and bandwidth. We want to put these resources to good use, to serve the needs of real-world financial businesses, ICOs and other organizations that want to build, test and deploy applications in a decentralized environment.

SWAZM DApps can be developed in any programming language that can run on top of the Linux microkernel. They can communicate and use existing available systems, not being forced to work with what's already in the blockchain.

SWAZM is changing the way DApps are created and accelerates the development lifecycle for blockchain applications. We do this by:
-allowing any existing programming language to be used inside its compute units
-control the environment where the application runs
-scale, debug and upgrade it on the fly
-select the geographical point where the compute units are deployed

5. SWAZM Architecture

5.1 SDTS (SWAZM Data Transfer Socket)

The goal of SDTS is to be limited only by hardware, disk or NIC bandwidth but not by latency. It's imperative to be highly efficient when it comes to CPU/IO resources utilization.

We are using only C programming language in order to have moderately structured and encapsulated code versus using high-level features of other languages. We are minimizing the number of system calls so we can maximize any systems throughput. While some treads are reading the others are writing and we are buffering data on both paths. This allows us to keep each subsystem busy while minimizing kernel switches.

For example, using our SDTS we were able to transfer data at a throttled 400 Mbps between high latency links Romania to Japan. That’s several times better than any of the speed of a highly optimized TCP/HTTP based solution and with a lot less strain on the system. When not throttling we were able to easily saturate a 1Gbps NIC and get almost to the theoretical link speed. Everything was optimized for commodity hardware with regular IO speeds and in-memory databases.

The role of SDTS in SWAZM is to provide a fast data transferring mechanism to serve as a virtual network adapter for any SWAZM node and DApps container in the SWAZM network. Key benefits:

- will allow the nodes to sync and spread data faster.
- will allow for quicker communication between nodes and outside traffic
- will empower the clustering of resources across all the SWAZM nodes

5.2 SDSL (SWAZM Data Storage Layer)

The goal of SDSL is to be a decentralized database and storage layer in order to handle large volumes of structured data across clusters of commodity servers. It organizes data and assures ACID transactions for all operations. It has excellent performance for write-intensive workloads and it's well suited for read/write workloads.

It was designed to provide SWAZM with a decentralized file system for its compute containers and provide Dapps with a place where they can work with persistent data. On top of this, is empowering SWAZM to be a decentralized sharded blockchain solution and perform transaction processing with high performance at an unprecedented scale.

5.3 Scalability

SDSL is supporting DApps which need a vast pool of performance requirements. By using a shared-nothing distributed architecture we scale up together with the network. The managing functions like data redundancy and partitioning caching are done automatically.

SDSL scales linearly with the number of CPUs in a cluster. For example, in a commodity hardware cluster, it scales to 8.2 million operations/sec performing a 90% read and 10% write workload with 16-byte keys and values between 8 and 100 bytes.

SDSL is offering a flexible solution, nodes can be provisioned and de-provisioned on the fly. As data is written on the storage layer, each piece of data is automatically placed on several nodes, without the interruption or degradation of the service. This kind of replication allows immediate load balancing based on the request and the data size.

5.4 Fault tolerance

In a decentralized system across many nodes, there is an increased probability of nodes crashing or migrating as the network gets bigger. That's why SDSL is designed to guarantee ACID properties even under the harshest conditions and much further than "no single point of failure".

In the case of failure, where multiple nodes are down, SDSL deals gracefully with the possible service loss. In this case, data unavailability is a valid use case because there are a finite number of replicas of each chunk of data. We are improving the odds of having data readily available by selecting packs of nodes on which to distribute the data. For example: in the case of 450 packs of 4 nodes the probability of data unavailability is reduced to approximately 0.48%.

5.5 Data Sharding and Replication

SDSL stores each piece of data on multiple nodes. If a node containing one of the copies is leaving the network, SDSL will automatically recover finding a new position for the lost copy. For the read operations, the communication is done directly to the nodes containing the replicas, checking a consistent view of the data.

5.6 ACID Transactions

A transaction is a set of reads and writes that is handled as a pack with some defining properties. All reads in a transaction are done upon the same state of the data layer and all the writes in a transaction either succeed or fail.

Everything that is read and written in SDSL is done through a transaction that is fully ACID (Atomic, Consistent, Isolated and Durable) and spanned across multiple nodes with high performance. On top of this, the isolation is the highest available, durability is the strongest and transactions are redundantly stored before they are considered committed.

5.7 CAP Theorem

The CAP theorem states that any data layer can provide strong consistency and system availability during network partitions. Even though the industry consensus is that this combination is impossible, we believe that this is a misunderstanding of the availability property.

Available in CAP means that all nodes are able to read and write even when partitioned. SDSL is ACID so during a partition we are choosing consistency over availability. This doesn't mean that the data is not available when some nodes are unable to communicate or to execute writes. We handle this properly through our fault-tolerant design because a network partitioning affecting a node is no different than a node going down.

The goal of SDSL is to keep the database and the application up, even if some nodes are down or are unable to communicate with the network. To achieve that, SDSL is configured with a set of consensus constraints (Paxos, Proof of Storage and Proof of Retrievability). SDSL is using a

shared state to maintain or update a replication topology so that in the case of a failure the nodes themselves update the replication topology.

5.8 Consensus

SDSL is using 3 types of consensus algorithms to move data around.

Paxos is a family of protocols for solving consensus in a network of unreliable processors. Consensus becomes difficult when the nodes and the communication network experience failures. The conditions that could prevent reaching consensus are difficult to predict, thus it's imperative to guarantee consistency.

Proof of Storage is a proof that in order to compute, the prover must use a specified amount of space, so it's clear that a certain node really has that available space around. Proofs of storage are cryptographic protocols that allow a client to efficiently verify the integrity of remotely stored data.

A proof of retrievability is a compact proof by a node (prover) to a certain client (verifier) that a target file is intact, in the sense that the client (the node retrieving the data) can fully recover it. Proof of retrievability incurs lower communication complexity than the transmission of the file itself.

The digital signature ensures the files integrity (proof of storage) and the ability to consume the file through a gateway (proof of retrievability).

5.9 Consistency

Consistency has 2 meanings in the storage layers, which are often confused:

The “C” in CAP refers to a consistency model, more exactly the conditions in which a write from a node is visible to the other nodes (for example, after 2 seconds you are completely sure that the data is consistent across all nodes).

The “C” in ACID refers to the property that data remains within the application integrity constraints. Usually, these are defined by their domain and can either be data values (“domain integrity”), a relation between data values (“referential integrity”) or a business rule from the application domain. Even though SDSL does not directly enforce them (they are defined by the application domain), the SDSL transactions with the guarantees of atomicity and isolation give the application the power to maintain the integrity as the application domain requires.

As a consistency model SDSL is using sequential consistency, providing the greatest ease of development.

5.10 Atomic operations over Data

SDSL implements a set of atomic operations over data without actually requiring the read back. This causes operations to be low latency and enables a vast range of data structures to be implemented efficiently as layers.

5.11 Load Balancing

SDSL moves chunks of data continuously from machine to machine, to balance the load every minute. Individual requests can be redirected from a busy node to a less busy one that has the replica of the data. These things work together in order to optimize latency and throughput.

5.12 Connection Pooling and Concurrency

SDSL is able to handle a very high number of concurrent connections because it doesn't create a thread per connection. This allows high performance even when we have 100k's of in-flight requests.

6. SCL (SWAZM Compute Layer)

SCL aims to specify the configuration, execution environment and life cycle of a container that is running on the SWAZM network. The configuration is specified in the swazm.json for the details and the fields that enable the creation of a container. The execution environment is specified to ensure that applications running inside a container have a consistent environment between runtimes (along with common actions defined for the container's lifecycle).

6.1 Principles of SCL

Let us define a SWAZM Container as a unit of software delivery. The goal of a SWAZM Container is to encapsulate a DApp and all the associated libraries in a format that is self-describing and portable. SCL will run it without extra dependencies, regardless of the environment where it runs.

The specifications for SWAZM Containers define:

1. a configuration file
2. a set of standard operations

A good analogy for the SWAZM Containers are the traditional containers used in the shipping industry. These are a fundamental unit of delivery, they are loaded, locked, stacked, unloaded and disposed of, irrespective of their contents. Standardizing the container itself, allows for a consistent, more streamlined and efficient set of processes to be defined and executed.

For SWAZM, containers offer a similar functionality by being the bedrock, standardized, unit of delivery for a Dapp.

6.2 Base Operations

SCL defines a set of Base Operations. They can be:

- created, started and stopped using standard container tools
- copied and snapshotted using SDSL
- downloaded and uploaded using SWAZM network tools

6.3 Content Independent

SWAZM containers are content independent:
-all basic operations have the same outcome no matter the contents
-they are started in the same way
-it's a Node application with the associated dependencies
-it's a Golang application that has a binary build in
-or any kind of application from an in-memory database or a bash script

6.4 Swazm Hosted Infrastructure

SWAZM containers run on the SWAZM infrastructure and can be bundled on your personal computer or infrastructure. On top of this, it can be uploaded to the SWAZM infrastructure using the SDSL tools, executed to a chosen SWAZM node and scaled to 30 SWAZM nodes across 30 countries (or cities, for example).

6.5 Designed for global scale

SWAZM containers are designed for automation at a global scale and they offer the same basic operations, regardless of the content. Many time-consuming things prone to errors can now be programmed and scaled effortlessly.

By the time a DApp is running, it had to be built, tested, and deployed. Multiple teams are involved and the entire process is slow, inefficient and expensive because it was dependent on the infrastructure constraints to a specific programing language.

6.6 Global scale Dapps delivery

SWAZM containers make the global scale of DApps a dream come true. Leveraging the principles listed above, SWAZM containers are removing programming languages barriers and scaling limitations or issues when developing DApps. SWAZM containers are changing the way the blockchain industry thinks about DApps development, delivery and scalability.

6.7 SWAZM Container format

A SWAZM container bundle encapsulates all the information needed to load and run a container: a set of structured files, all the necessary data and metadata for SCL to perform the basic operations needed for compute.

The constraints at the container level are only referring to dependencies, data stored on a regular file system and bundled so it can be executed by the SCL.

This includes the following artefacts:

1. the file must be named swazm.json, contain the configuration data for a SWAZM container and must reside in the root of the filesystem of the bundle
2. SWAZM containers encapsulate all the DApp executables, the metadata, the configuration files and anything else that the DApp is relying on

A tar archive of a SWAZM container bundle will have these artefacts at the root of the archive.

7. SWAZM Container Runtime and Lifecycle

7.1 State

The state of a SWAZM container includes the following attributes:

1. The identifier which is unique across the network
2. The runtime status of a SWAZM container
    - Deploying - the SWAZM container is being deployed on the network
    - Deployed - the SCL found a node on the SWAZM network and it has been deployed to that specific node
    - Running - the SWAZM container is executing the DApp bundle but it hasn't presented any exit code
    - Stopped - the SWAZM container has received a specific exit code and has stopped

3. The identifier in the SDSL (where the bundle is stored) is provided so developers can identify a DApp by it's bundle version.

7.2 Life cycle

The life cycle describes the order of events from when a SWAZM container is deploying until the moment it has stopped.

1. Deploy command is invoked with the unique identifier of the SDSL storage where the DApp bundle is located.
2. The SWAZM container environment is created according to the swazm.json runtime environment. If the creation process fails, it will generate an error.
3. Start command is executed with the unique identifier of the SWAZM container.
4. Prestart hooks are executed, if they fail an error is generated.
5. The SWAZM container runtime executes the DApp specific program.
6. The post start hooks are invoked by the runtime and if they fail a warning is generated but the execution continues.
7. The SWAZM container exists, it can happen during an erroring out, exiting, crashing or the kill command being invoked in the runtime.
8. The runtime executes the delete command on the unique identifier of the SWAZM container.
9. The SWAZM container is destroyed by reverting the changes that were executed during the deploy phase.
10. The post stop hooks are invoked and if anything fails a warning is logged but the process continues.

7.3 Proof of Verifiable Computing

Gennaro et al. defined the notion of verifiable computation scheme as a protocol between two polynomial time parties to collaborate on the computation of a function $F: \{0,1\}^n \rightarrow \{0,1\}^m$. This scheme consists of three main phases:

Preprocessing. This stage is performed once by the client in order to calculate some auxiliary information associated with F. Part of this information is public to be shared with the worker while the rest is private and kept with the client.

Input preparation. In this stage, the client calculates some auxiliary information about the input of the function. Part of this information is public while the rest is private and kept with the client. The public information is sent to the worker to compute F on the input data

Output computation and verification. In this stage, the worker uses the public information associated with the function F and the input, which are calculated in the previous two phases, to

compute an encoded output of the function F on the provided input. This result is then returned to the client to verify its correctness by computing the actual value of the output by decoding the result returned by the worker using the private information calculated in the previous phases.

The defined notion of verifiable computation scheme minimizes the interaction between the client and the worker into exactly two messages, where a single message is sent from each party to the other party during the different phases of the protocol.

To instill greater confidence in computations outsourced to the cloud, clients should be able to verify the correctness of the results returned. To this end, we are using Pinocchio, a system built for efficiently verifying general computations while relying only on cryptographic assumptions.

With Pinocchio, the vault creates a public evaluation key to describe its computation; this setup is proportional to evaluating the computation once.

The SCL then evaluates the computation on a particular input and uses the evaluation key to produce a proof of correctness. The proof is only 288 bytes, regardless of the computation performed or the size of the inputs and outputs. Anyone can use a public verification key to check the proof.

7.4 SVPS (SWAZM Vault Permission System)

In SVPS a vault is an entity that issues digital certificates. A digital certificate certifies the ownership of a public key by the named subject of the certificate. This allows others (third parties such as developers or ephemeral contractors) to rely upon signatures or on assertions made about the private key that corresponds to the certified public key.

A vault acts as a trusted third party, trusted both by the subject (owner) of the certificate and by the party relying upon the certificate. The format of these certificates is specified by the X.509 standard.

SWAZM has a particularly common use for a vault, it allows entities to authenticate the recipient of the certificate. The techniques used for vault validation vary, but in general vault validation techniques are meant to prove that the certificate applicant controls a given stake in a certain DApp application domain, but doesn't have management control over the vault or the DApp domain. It’s mostly used by a vault manager as a control tool to protect the resources in the vault. For example, as a vault owner, you may want to revoke rights to anyone that might use your DApp domain while having the flexibility to allow ephemeral access to a pool of SWAZM resources.

7.5 Issuing Vault Certificates

A vault issues digital certificates that contain a public key and the identity of the owner. The matching private key is not made available publicly, but kept secret by the end user who generated the key pair.

The certificate is also a confirmation or validation by the vault that the public key contained in the certificate belongs to the person, organization, server or other entity noted in the certificate. A vault's obligation in such schemes is to verify an applicant's credentials, so that users and relying parties can trust the information in the vault's certificates. Vaults use a variety of standards and tests to do so.

In essence, the certificate authority is responsible for saying "yes, this person is who they say they are, and we, the vault, certify that".

7.6 Revoking Vault Certificates

A vault revocation list is a list of certificates that have been revoked by the issuing vault before their scheduled expiration date and should not be trusted anymore.

- A vault revocation list is generated and published periodically, often at a defined interval.
- A vault revocation list can also be updated immediately after a certificate has been revoked.
- A vault revocation list is issued by the vault operator which also issued the corresponding certificates.

Expiration dates are not a substitute for a vault revocation list. While all expired certificates are considered invalid, not all unexpired certificates should be valid. Vault revocation lists or other certificate validation techniques are a necessary part of the SWAZM infrastructure, as mistakes in certificate vetting and key management are expected to occur in real-world operations.

There are two states of a revocation certificate:

1. **Revoked**: A certificate is irreversibly revoked if, for example, a private-key is thought to have been compromised. Certificates may also be revoked for failure of the identified entity to adhere to policy requirements or violation of any other policy specified by the vault operator. The most common reason for revocation is that the user is no longer in sole possession of the private key (e.g., the token containing the private key has been lost or stolen).

2. **Hold**: This reversible status can be used to note the temporary invalidity of the certificate (e.g., if the user is unsure if the private key has been lost). In this example, if the private

key was found and nobody had access to it, the status could be reinstated, and the certificate is valid again, thus removing the certificate from the vault revocation list.

7.7 Inter Blockchain Communication

The SWAZM infrastructure is designed to support inter-blockchain communications. This is achieved by special endpoints inside the SCL and presenting DApps developers specific endpoints where they communicate with the other blockchains.
When communicating with another outside blockchain, block producers must wait until they are absolutely certain that a transaction has been irreversibly confirmed by the other blockchain before accepting it as a valid input.

SCL will also use Merkle trees to track transaction history, while is pretty hard to catch the pending blocks, there is an absolute certainty when verifying offline the blocks from a block producer.

The SCL supports agnostic languages, so if a certain DApp needs a full node from a certain blockchain, it can be easily incorporated into a SWAZM container and executed on the network.

8. Applications

In general, any type of applications can run on SWAZM, from DApps that are in the stage of raising money in order to decentralize a piece of the market (they need decentralized compute platform) to traditional applications that are using traditional databases.

8.1 Exchanges

Due to the fact that SWAZM has support for interoperability, an exchange would have several advantages to be built on SWAZM:
-transparency
-decentralized nature
-community - it would be easier for developers to contribute, develop its wallets or the token mechanics

8.2 DApps ICO

There are numerous initiatives out there that hold vast promises into disrupting traditional technologies through the use of blockchain. Even though most of them are lacking the technology to achieve that, by taking advantage of SWAZMs decentralized storage and compute layers, they can easily develop the technology using the already existing programming

languages and technologies. They would only be focused on what makes their product unique and leverage the SWAZM technology to deliver on that promise and reach their target market.

8.3 Existing Hosting Providers

By using SWAZM, existing hosting providers can leverage the power of their unused bandwidth/compute/storage infrastructure. By joining forces they would be able to monetize their resources in a democratic manner inside SWAZM and be part of a competitive infrastructure which can take on the biggest established players on the market.

8.4 Storage Provider Applications

By leveraging the technologies that SWAZM offers, like fast transfer technologies and decentralized storage, the community could build numerous applications that may challenge the traditional storage providers.

8.5 Content Delivery Networks

By leveraging the global city level distribution of SWAZM nodes, a content delivery network could leverage high speeds and targeted content storage for web assets, media or any kind of network. For example, there have been cases where huge media providers were forced to pay fees to ISP because the household connections were not performant enough to deliver the content in specific areas. SWAZM solves such cases by delivering peer to peer content through it’s storage layer.