1. Introduction

Previously, we built a distributed file sharing system 'EZShare' based on an asynchronous request-reply model between servers and clients. The system provides the functionalities of publishing a public resource (PUBLISH), sharing a resource on server (SHARE), removing an existing resource (REMOVE), querying a resource by various attributes across servers (QUERY), fetching a shared resource (FETCH) and informing the server of a list of other servers (EXCHANGE).

This time, we further mounted the security feature by enabling secured connections between clients and servers, and subscription functions by implementing long-time connections with asynchronous conversations.

In the report, we elaborated the mechanisms adopted to realize the two additional features, discussed the effectiveness of these mechanisms, proposed alternatives to address the issues, and compared the advantages and disadvantages between different solutions.

2. Security Mechanisms

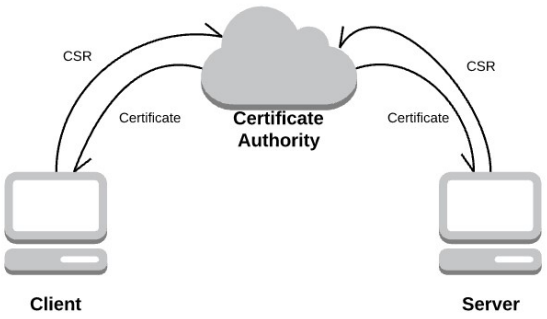
2.1 Security Threats Identification

There were several potential security threats in original implementation of EZShare system. ***One*** of them was information leakage, which meant the private information that should only be available to specific clients was captured by an illegimate party. There were two possible approaches to accomplish such attacks, which could use Man-in-the-Middle (MITM) and fraudulent client/server separately. ***Another*** threat was unauthorized tampering of information, which meant the information provided to the servers or clients was tampered and no longer correct. This could also be achieved by MITM attack. ***Additionally***, attacks such as Denial of Server (DoS) were also severe threats we were faced with.

2.2 Security Mechanism in EZShare

To address the security issues mentioned above, we implemented Secure Socket Layer (SSL) Protocol in the system and controlled the querying frequency of the same IP address on server side.

To build SSL Protocol, we established key stores on both client and server side. Inside the key stores, there are keys and well-known certificates. We then generated the certificate signing requests from both sides and received certificates signed by Certificate Authority (CA). Since client and server each had a certificate signed by a trusted third-party, we imported the root certificate and their certificates into key stores. And client and server could establish connection and conversation based on SSL protocol.



Graph 2.2.1 Requests for Signed Certificates

When a client tried to establish connections with a server in a secured manner, both the client and server should check each other's identity and ensure the other side was a trusted entity in their own key stores. After the handshake procedures, they would negotiate the encryption method and a secret used for generating the session key, and then started the secured communication with the shared session key.

In addition to SSL Protocol, we also implemented a mechanism to prevent the same IP address from sending requests too frequently. For

example, if a client sends multiple requests (equals to or more than three times) within a short interval, the server would close the subsequent requests from that IP address without processing the requests.

2.3 Performance Assessment

In general, the security mechanism adopted by EZShare solved the security issues to some extent. ***First of all***, the SSL Protocol required the client and server to verify each other's identity, and encrypted the data transmitted on the secured channel. This implementation mitigated the risk of information leakage. ***Secondly***, since the session key was negotiated and the data transmission was encrypted, it would be hard to perform MITM attack during the conversation. So, the threat of information tampering could be reduced. ***Thirdly***, the mechanism that prevented the same IP address from frequently sending request could relieve the threat of DoS attack to some degree.

However, there were many other security threats that could not be addressed by the implemented mechanisms.

Above all, we still reserved insecure ports and channels, which are sensitive to information leakage and tampering through unauthorized accessing. For instance, the attacker could easily perform eavesdropping to get information, perform tampering to change data, or fake a client to forge message.

Then, for secured sockets, there were some weakness for MITM attack during the handshake process. For one thing, the client and server both followed the certificate chain when deciding whether to trust an entity. If the root certificate was trusted, then all its subsidiaries should also be trusted. But in reality, it was questionable whether all the certificates signed by root CA should be equally trusted. For another, there were MITM attacks aiming at SSL handshake process such as SSL Freak.

Finally, regarding DoS attack, although we had implemented the mechanism to prevent single IP address from accessing the server too frequently, we could not survive the situation where the attacker faked different IP addresses to perform the attack or the attacker manipulated multiple hosts to perform the Distributed Denial of Service (DDoS) attack.

2.4 Alternative Security Mechanisms and Tradeoffs

Enhancing security on the system may always have some tradeoffs like sacrificing the applicability of certain functionalities. Thus, it is hard but meaningful to select the security mechanisms that best suits the system functions.

Currently, we hide the owner information of the resources from query request in order to protect the sensitive data. However, this implementation is not convenient for client to check their own resources and limits the extendibility of the system.

An alternative mechanism for protecting sensitive data is that we may encrypt the sensitive fields such as owner and channel with symmetric algorithm between user and server. To generate the shared key on a per-user basis, we first need to build the register and login function. Then, for each registered user, the system will generate a shared key for them. In this way, the server can ensure that only the user who published or shared the resources can decrypt and see the owner and channel information of the resources.

We suggest symmetric algorithm instead of asymmetric algorithm for two reasons: one is that we assume that the conversation between different entities in the system have been established on SSL and have been secured to some extent; and the other is that symmetric

algorithm requires less computing power and processes faster when encrypting and decrypting messages.

An obvious pro of this mechanism is that it will enable the functionality that has been limited by security concerns. Users can see the owner and channel information of the resources which have been published by them. Nevertheless, the con of this mechanism is that more computing power will be required for encrypting and decrypting. And there are still security threats even if we implement this mechanism on the system.

3. Subscription

3.1 Relay Mechanism between Servers

In EZShare, servers need to relay the subscription to other servers when client set the RELAY parameter to be true. We implemented this function in a straight-forward way.

Generally, a server relayed the subscriptions to other servers by creating new threads and initiating persistent connections. When new resources were published and shared on another server, that server would send back the resources satisfying the subscription requirements. Then, the server received the resources from the connection and sent them back to the client.

How did each server distinguish new resources from old ones? We solved this problem by adding a timestamp for each resource at the time the resource was published or shared. At the same time, we put another variable of timestamp inside the subscription function, which represents for the most recent time we checked the resource list for new resources. In this case, we could check for new resources periodically in a short interval.

3.2 Performance Assessment

Although we achieved the function of relaying a subscription to other servers, the strategy we adopted consumed lots of resources (e.g. threads and bandwidth) on all the servers. This highly constrained the scalability of our distributed system, and reduced the performance of system under high traffic load. What's more, this implementation was sensitive to single node failure, because if one node was disconnected from the system, the published resources could no longer be obtained by subscriptions on other servers.

But meanwhile, the advantage of implementing persistent connections was that the client could receive the subscripted content in real time. Whenever there was a new feed on another server, the initial server would receive a notification quickly, and replied to client without much delay.

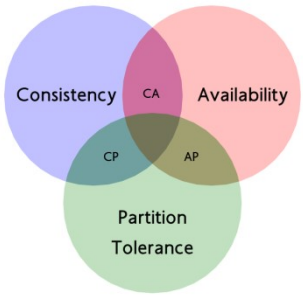
3.3 Alternative Mechanisms and Comparisons

An alternative mechanism is that instead of adopting a persistent connection between servers, one server can periodically check another server for any updates with a temporary synchronous connection. For instance, in every 10 minutes, one server can establish a connection with another server and send a polling request. Through the request, the former server will get the latest resources published or shared on the latter one during the interval. Then, the former server can check the resources and forward them to clients who subscribe certain resources.

On one hand, there are several advantages of implementing this mechanism compared to the persistent asynchronous one. The most prominent feature is that this mechanism ensures the scalability and performance of system by saving the resources used for persistent connections. In a distributed system with multiple servers, if one subscription requires hundreds of threads to relay the requests to many other servers on persistent connections, all servers will

quickly run out of threads and other resources, and provide terrible services to rest of the clients.

Another advantage of this mechanism is that it can ensure the availability and eventual consistency of subscription even under partition situations. According to CAP theorem, we cannot enjoy the availability, consistency, and partition tolerance at the same time. For example, if the a server is disconnected from other servers and still provides services to clients, the subscripted content it sends back to clients is not consistent temporarily, because once the server connects to other servers, it will continue to perform periodically polling, and obtain the updated resources since last check. Then, the subscription will achieve 'eventual consistency'.



Graph 3.3.1 CAP Theorem

However, **on the other hand**, there are disadvantages of applying non-persistent connections for relaying subscriptions. Since the server could only update the resources after periodically checking another server, the notification send to client is not in real time. Besides, if the checking interval is set too short, the frequent polling will also consume the resources on servers.

4. Conclusion

To sum up, we implemented two more features on previous EZShare System, the security feature and the subscription feature. We introduced the details of the mechanisms, discussed potential alternatives, and assessed the performance by comparing different mechanisms.

However, we concluded that no matter which mechanism to choose, we should also balance the tradeoffs between different options.

References

[1] Lin, C., & Varadharajan, V. (2006, April). Trust based risk management for distributed system security-a new approach. In *Availability, Reliability and Security, 2006. ARES 2006. The First International Conference on* (pp. 8-pp). IEEE.

[2] Brewer, E. (2010, July). A certain freedom: thoughts on the CAP theorem. In *Proceedings of the 29th ACM SIGACT-SIGOPS symposium on Principles of distributed computing* (pp. 335-335). ACM.

[3] Aaron, H., & Shanika K. (2017, Febrary). Distributed Systems Lecture Notes.