Presence Scalability Architectures[1]

Vishal K. Singh and Henning Schulzrinne
Department of Computer Science, Columbia University
{*vs2140, hgs*}@cs.columbia.edu

Abstract: We apply the two-stage reliable and scalable SIP server architecture proposed in [1] for presence. The first stage proxies the requests to one of second stage server groups based on the event header and destination user identifier. The destination user identifier is based on presentity's URI. Such a system achieves uniform load sharing on the servers on an average. However, in certain cases the load may not be uniformly distributed on all the servers. We propose to use load metric based static allocation algorithm to distribute the load uniformly. The load metric determination and performance evaluation of such a strategy is identified as future work. Additionally, we explain load sharing architecture for XCAP server, which is based on HTTP request redirection which is similar to load sharing in web servers.

1. Introduction

The scalability of presence [5] system becomes increasingly important with growing number of presence based applications e.g., instant messaging, wireless and wireline networked applications. It is important to know when a new server needs to be added to the presence system. Additionally, the system capacity must increase linearly with additional hardware i.e., CPU and memory.

This short paper is an extension of [1] for load sharing of presence server. We apply the two-stage scalable Session Initiation Protocol (SIP) [7] server architecture proposed in [1] for the case of presence. The technique is transparent to the client application. However, presence system is also an event processing and notification system. The incoming requests can be classified based on user identifiers as well as event package [10]. Each event type is processed differently by the server. The load sharing architecture proposed in [1] can be extended to handle multiple event types using one more level of indirection, which distributes request based on the type of event. This can be done in the first stage servers itself or by adding one more stage which does request forwarding based on event type. Additionally, presence server involves complex XML processing and can result in multiple notifications to be generated for each received PUBLISH [11] message, which is different from normal SIP based call processing server.

The remainder of this document is organized as follows: Section 2 presents an overview of presence server and presence data processing, Section 3 explains factors affecting the scalability of presence server. Section 4 presents the load sharing architecture, section 4.1 explains the two stage load sharing architecture proposed in [1], section 4.2 introduces a load balance metric and identifies future work for load sharing in presence based on load balance metric, section 4.3 explains load sharing for XCAP [17] server. In section 5, we present the evaluation strategy for load sharing mechanisms using SIMPLEStone [16]

[1] This work is supported by Verizon labs.

benchmarking standard. Then we present some future works in section 6, conclusion in section 7 and provide the references in section 8.

2. Presence Overview and Data Processing

A presence system allows for users to subscribe to each others presence [5] (availability and willingness for communication) information. The users (Watchers) subscribe to presence information of other users (Presentity) using SIP SUBSCRIBE [10] and are notified about the changes in state of other users by SIP NOTIFY [10] messages. Presence data for a user (Presentity) is published from different presence sources using SIP PUBLISH. The received presence data is processed and distributed to give to the watchers a consistent view of the status of the presentities they are interested. Fig. 1 shows a basic block diagram of presence system.

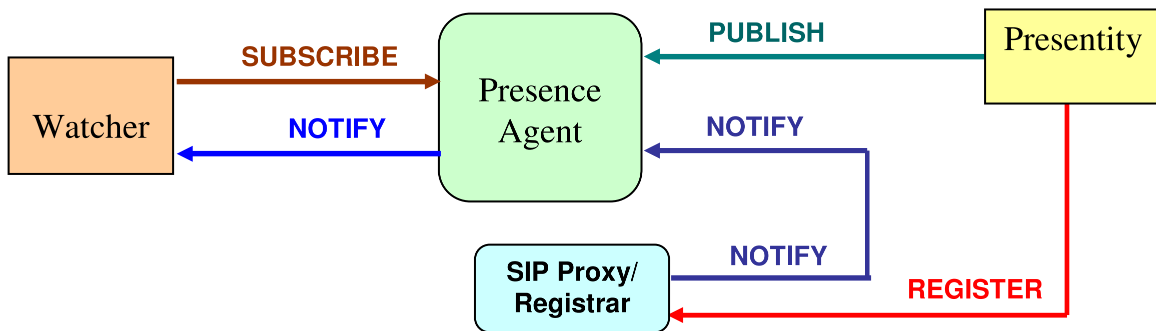

Fig.1. Basic block diagram of presence system

Figure 2 shows the processing on presence server. This is explained in detail in SIMPLEStone in section 2.1

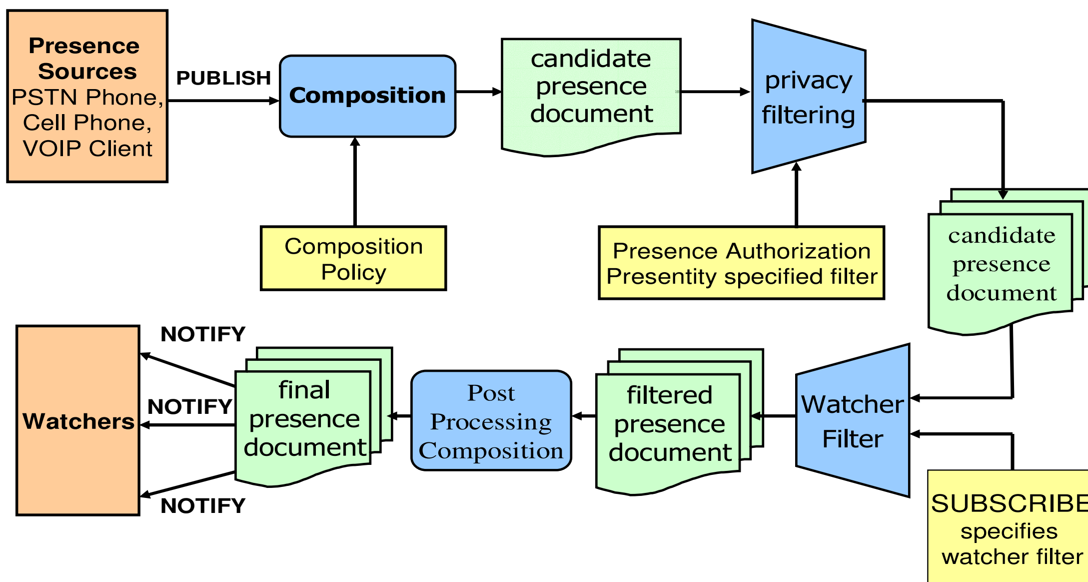

Fig. 2 Presence processing overview

3. Factors Affecting Presence Server Scalability

In this section, we explain different factors which affect the presence server scalability.

Request Rate: Number of messages received and distributed by the presence server per second. The number of PUBLISH messages received depends on average number of sources and their PUBLISH rates. Similarly, the number of SUBSCRIBE messages received and NOTIFY messages sent is a factor of number of watchers for each presentity.

Filter document size or Number of rules:
Every notification is generated after performing the rule matching process and applying the matched filter rules. This implies checking conditions and applying actions and transformations as specified in common policy draft [18] and in presence authorization rules [12] draft. The processing of this step depends on the number of rules in the policy document of the presentity.

Composition
The type of composition [14] policies that the server supports and application of the policy, i.e., different policy being used on a per presentity basis or applied globally on the server for all presentities. This determines the processing done by the server and hence affects the scalability. Composition can also affect the size of PIDF [6] document. An intelligent composition based on a rule language will load the presence server higher than the default composition based on union or overriding policy.

Watcher filtering
The size of watcher filter [13] sent by the watcher in SUBSCRIBE message affects both the processing and amount of traffic generated.

Partial notification
Partial notification [22] is mechanism used to conserve bandwidth by sending only the changes in the presence document to the watchers. The watchers generate the complete event state from the partial presence documents. The server compares the updated document with the old document for the presentities and generates the partial presence document.

Transport: The transport protocol used like TCP, UDP, or TLS affects the performance of the server.

Other factors: Other factors that can affect the scalability are DNS look up, XCAP change event handling, database optimizations, database vs. in memory design etc. Each of these contributes to determine the load on system.

4. Load Sharing Architectures

In this section, we explain the two stage identifier based scalable load sharing architecture for presence. We also explain the load balance metric based load sharing architecture and HTTP redirect based load sharing architecture for XCAP.

4.1 Two-Stage Identifier based architecture

As explained in [1] the presentity identifier space is divided into non-overlapping groups. A hash function maps the identifier to a particular server group that manages presentity's presence information. For Example, The first stage server (P0 or P0') proxies the presence requests to P1or P2 based on the destination user identifier. For example, when a PUBLISH or SUBSCRIBE is received for bob@a1.com and H (bob) is 1 then it goes to P1, whereas sam@a1.com where H (sam) is 2 goes to P2. To guarantee almost uniform distribution of presence requests to different servers, a better hashing algorithm such as SHA1 can be used or the groups can be re-assigned dynamically based on the load. The first stage server is selected based on DNS SRV [19] and NAPTR [20] records.

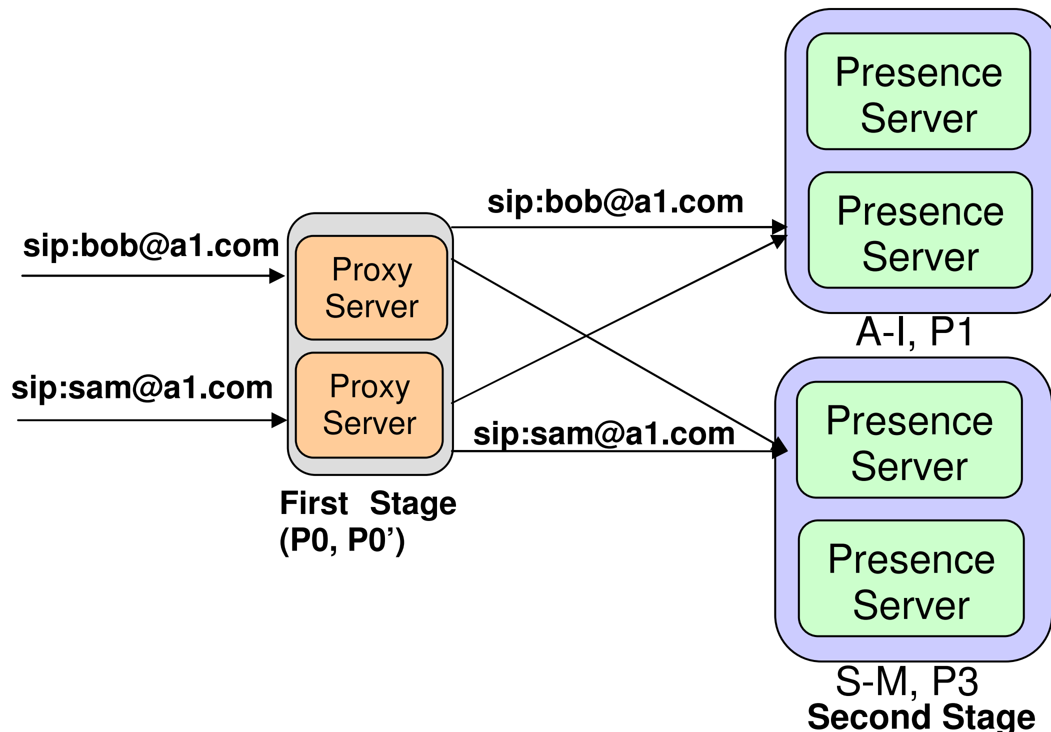

Fig 3.Two stage scalable load sharing

This architecture is explained in much detail in [1]. We explained above how PUBLISH and SUBSCRIBE are sent to the appropriate presence server. SIP SUBSCRIBE and PUBLISH allow multiple event types including presence. Thus, we can add one more stage of proxy servers to distribute these requests based on type of event. , Adding a new stage will be similar to the first stage shown in Figure 3 and would map event types to one of the first stage server groups. Optionally, we can add event based forwarding in the first stage itself. In which case, the first stage chooses a server pool based on event type and choose the server from the pool based on hash of user identifier.

4.2 Load Metric based architecture

Each presence request may generate different amount of load on the server depending on the number of filter rules, the PIDF document size, the number of watchers of the presentity etc. The load sharing mechanism assumes that on an average the request rates, filter sizes, etc., are randomly distributed such that the hash function generates uniform distribution to share the load more or less equally among the second stage servers. However, this may not be the case always e.g., a set of presentities allocated to a server generating high request rate, or having huge filter sizes. To address such scenarios we propose to use load metric (LM). Load metric is the load generated by a presentity on a server. The sum of load metrics for all presentities gives the total load. Using this metric we can statically allocate presentities to different server groups. This allocation is provided to the first stage servers so that it can forward the requests to the correct server group. LM can be calculated as a sum of load generated by each operation like privacy filtering (filter size), watcher filtering, composition (policy per presentity), sending notification to the watchers etc. LM also depends on the rate of PUBLISH and SUBSCRIBE for the presentity. An example of load metric calculation is given below

Total Load = num of presentity X [load metric for presentity]

= num of presentity X load [(composition) * 1 + (privacy filtering + watcher filtering + notify generation) * num of watchers] + C.

Where C is load added for other factors like transport protocol, design choices like DB vs. in-memory etc.

However, quantifying load for each operation to measure LM is an open issue. One mechanism is calculating LM is by determining the load for each operation experimentally.

4.3 XCAP Server Load balancing Architecture

Load balancing XCAP server uses the same architecture as is used for load balancing web servers. For such architecture to work for creating and updating policy files, buddy lists, resource lists etc., the XCAP servers need to have access to user's information to authenticate the requests and access to the user's documents.

The following load balancing mechanisms can be used.

A) DNS based load balancing

The DNS SRV [19] and NAPTR [20] mechanisms can be used for load sharing using priority and weight fields in the resource records. It can be used to statically distribute load in proportions of existing load on the servers. Additionally, DNS can be used to send requests in round robin fashion.

B) HTTP Redirect request based load balancing.

In this mechanism, the first stage HTTP redirect server sends HTTP redirect [21] responses with the address of the XCAP server to the clients. The client can then directly connect to the XCAP server. The redirection server can be based on existing load on the servers, round robin or using a hashing algorithm where a range of identifiers are

statically allocated to the second stage servers. If we want to do round robin or load based redirection, all the servers need to have access to users credentials and policy files, so that the requests can be authenticated and their policy files be updated. Figure 4 shows two-stages of load balancing server for XCAP requests where requests can be served by any of the second stage servers. The XCAP servers use a directory service e.g., mapping the request URI to file system to get the XML documents. The second request directly goes to XCAP server.

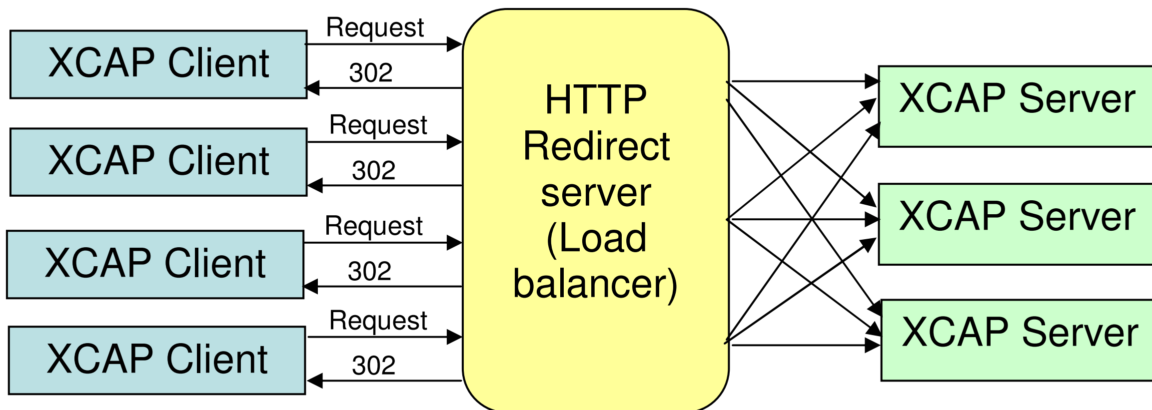

Fig 4 HTTP redirect requests based load balancing.

C) There is other techniques to achieve load sharing, e.g., Packet rewriting, TCP splicing techniques, based on how this goes. These mechanisms are based on additional hardware to re-encapsulate packets at wire speed. They are limited by the throughput of the switch being used to do that.

5. Performance Evaluation Strategy

The scalability scheme can be evaluated using the SIMPLEStone specification. The loader and handler tools can be used to perform test runs, with increasing load levels, targeted at the server set up being tested. The load is increased till the server starts dropping requests and the additional server can be added. We can verify if the request rate scales linearly with number of servers. This can be repeated with different hashing and presentity allocation mechanisms.

6. Future Work

We plan to do the scalability and performance testing using the two stage architecture. The additional indirection for event type in the three stage architecture also needs to be validated when additional events type need to be supported. Further, we would like to test the presentity migration using a load metric based static allocation algorithm. The load metric for each of the factors need to be determined experimentally and should be used to determine the load generated per presentity.

7. Conclusion

We explained various factors which limit the scalability of a presence system. We explained the two-tiered architecture and explained how to use it in the context of

presence. We explained load balance metric based solution for certain deployments. Additionally, we presented load sharing architecture for XCAP server.

8. References

1. Kundan Singh and Henning Schulzrinne, "Failover and Load Sharing in SIP Telephony", International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS), Philadelphia, PA, July 2005.
2. Rosenberg, J., "A Data Model for Presence"draft-ietf-simple-presence-data-model-07, January 22, 2006.
3. Rosenberg, J., "A Processing Model for Presence" draft-rosenberg-simple-presence-processing-model-01, July 17, 2005.
4. Lonnfors, M., "Session Initiation Protocol (SIP) extension for Partial Notification of Presence Information" draft-ietf-simple-partial-notify-05, May 24, 2005.
5. Day, M., Rosenberg, J. and H. Sugano, "A Model for Presence and Instant Messaging", RFC 2778, February 2000.
6. Sugano, H., Fujimoto, S., Klyne, G., Bateman, A., Carr, W. and J. Peterson, "Presence Information Data Format (PIDF)", RFC 3863, August 2004.
7. Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M. and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, June 2002.
8. Rosenberg, J., "A Presence Event Package for the Session Initiation Protocol (SIP)", RFC 3856, August 2004.
9. Schulzrinne, H., Gurbani, V., Kyzivat, P. and J. Rosenberg, "RPID: Rich Presence: Extensions to the Presence Information Data Format (PIDF)" draft-ietf-simple-rpid-08, July 16, 2005.
10. Roach, A., "Session Initiation Protocol (SIP)-Specific Event Notification", RFC 3265, June 2002.
11. Niemi, A., “Session Initiation Protocol (SIP) Extension for Event State Publication ", RFC 3903, October 2004.
12. Rosenberg, J., "Presence Authorization Rules" draft-ietf-simple-presence-rules-04, October 18, 2005.
13. Khartabil, H., "An Extensible Markup Language (XML) Based Format for Event Notification Filtering" draft-ietf-simple-filter-format-05.txt, March 15, 2005.
14. Schulzrinne, H., “Composing Presence Information” draft-schulzrinne-simple-composition-00, July 10, 2005
15. Schulzrinne, H., Sankaran Narayanan, Jonathan Lennox and Michael Doyle, “SIPstone - Benchmarking SIP Server Performance”.
16. Singh, V., Schulzrinne, “SIMPLEStone - Benchmarking presence Server Performance”.
17. Rosenberg, J., "The Extensible Markup Language (XML) Configuration Access Protocol (XCAP)", draft draft-ietf-simple-xcap-08, Oct 24, 2005.
18. Schulzrinne, H., "A Document Format for Expressing Privacy Preferences", draft-ietf-geopriv-common-policy-06, Oct 2005.
19. A. Gulbrandsen, P. Vixie, and L. Esibov, “A DNS RR for specifying the location of services (DNS SRV)," RFC 2782, Internet Engineering Task Force, Feb. 2000.

20. M. Mealling and R. W. Daniel, “The naming authority pointer (NAPTR) DNS resource record," RFC 2915, Inter-net Engineering Task Force, Sept. 2000.
21. Fielding, R., Gettys, J., Mogul, J., Frysyk, H., Masinter, L., Leach, P. and T. Berners-Lee, "Hypertext Transfer Protocol HTTP/1.1", RFC 2616, June 1999.
22. Lonnfors, M., Costa-Requena, J., Leppanen, E., Khartabil, H., Session Initiation Protocol (SIP) extension for Partial Notification of Presence Information, draft-ietf-simple-partial-notify-06, October 21, 2005.