Järvinen, Juha; Marttinen, Aleksi; Luoma, Marko; Peuhkuri, Markus; Manner, Jukka

Architecture of XMPP proxy for server-to-server connections

Published in:
2017 Military Communications and Information Systems Conference (MilCIS)

DOI:
10.1109/MilCIS.2017.8190423

Published: 14/12/2017

Document Version
Peer-reviewed accepted author manuscript, also known as Final accepted manuscript or Post-print

Please cite the original version:
Järvinen, J., Marttinen, A., Luoma, M., Peuhkuri, M., & Manner, J. (2017). Architecture of XMPP proxy for server-to-server connections. In *2017 Military Communications and Information Systems Conference (MilCIS)* IEEE. https://doi.org/10.1109/MilCIS.2017.8190423

Architecture of XMPP Proxy for Server-To-Server Connections

Juha Järvinen, Aleksi Marttinen, Marko Luoma, Markus Peuhkuri and Jukka Manner
Critical Infrastructure Research Unit, Aalto University
Postal Address: PO Box 15600, FI-00076 AALTO
Email: {Juha.Tapio.Jarvinen, Aleksi.Marttinen, Marko.Luoma, Markus.Peuhkuri, Jukka.Manner}@aalto.fi

***Abstract*—The Extensible Messaging and Presence Protocol (XMPP) is one of the most popular Instant Messaging (IM) protocols which uses a client-server working mode. This protocol uses different connection primitives for both client-to-server (c2s) and server-to-server (s2s) connections. It is actively used in mission-critical operations where the reliability and security of communication systems is always imperative. One approach to secure services and private networks is to use proxy services as security gateways. Proxies enable interoperability between different security domains acting as Information Exchange Gateways (IEGs).**

In this paper we present an architecture of the XMPP proxy for s2s connections. The system is based on an Openfire XMPP server with a Hazelcast clustering plugin, and a Hazelcast clustering link is used between the XMPP server and the XMPP Proxy. We have constructed an implementation to verify and validate the presented approach. Our proposal enables an effective seamless connection for XMPP proxies. Furthermore, it increases the system security for example, terminating both TCP and XMPP flows to prevent malicious attacks. Finally, we show that the proposal does not significantly increase the anticipated delay of the communication.

I. Introduction

In the early days of the Internet, security had not yet become a major issue. However, after numerous malpractices, viruses, and attacks, times have changed. Hence, nowadays a significant part of communication engineering is the consideration of security issues. Meanwhile new communication protocols are continuously being presented, and old ones are being used in novel ways, placing security systems in difficulties. The challenge is the detection of desirable and malicious traffic. Typically, security is improved in communication networks, by using with different devices, for example, firewalls or proxies, depending on the particular use case.

The requirement for fast and effective communication between computers is continuously on the increase. For the computer-aided communications between humans, several protocols and applications have been proposed. One of the most popular of such has been Internet Relay Chat (IRC) [1]. By default, however, the security of IRC is low, and it should not be used in mission critical-communications. More advanced and also suitable for machine-to-machine (M2M) communications is the Extensible Messaging and Presence Protocol (XMPP) [2]. As an additional features to IRC, the XMPP offers presence information notifications and security features. It is is a federating Instant Messaging (IM) protocol and carries out a client-server communication method where client-to-server (c2s) and server-to-server (s2s) methods differ from each other. Both the c2s and s2s connections are one-way flows in the XMPP, $i.e.$ both directions have to be established separately.

Instant messaging is one of the key communication services in military operations. The challenges of fast messaging have been extensively tackled in several previous and ongoing military research projects. For instance, both U.S. DoD (Department and Defence) and NATO (North Atlantic Treaty Organization) are using widely the XMPP protocol in instant messaging. However, if XMPP services, servers, and flows are not properly secured, it is easy for adversaries are enabled to significantly disturb the communication, which may even compromise the success of missions.

In recent years several scientific articles have discussed the usage of instant messaging services in military communications. Lass $et\ al.$ proposed several extensions to the XMPP protocol in [3]. Their approach included, for instance, a serverless method of XMPP. Furthermore, they discussed the utilization of the gateway and proxying components of the XMPP. The proposals of the paper is based on their earlier work on the GUMP (Generic Unicast-to-Multicast) proxy. [4] Skjegstad $et\ al.$ proposed a mechanism to improve the message delivery rate of the XMPP in tactical MANETs (Mobile Ad hoc Networks). Their proposal also enables serverless communications, which is highly useful in the tactical MANETs where connections may be very unreliable due to highly dynamic environments. [5]

The contribution of this paper is a novel architecture to improve the security of XMPP instant messaging systems. The presented architecture separates the direct s2s connection from the Mission Network (MN). Our research scenario is related to mission networks, however the proposed architecture is also current over more vicious public Internet connections. The architecture enables attaching numerous nodes in the XMPP clusters in the name of load sharing, enabling seamless internal communication even during the denial-of-service (DoS) attacks. Under a DoS attack an internal XMPP system is functioning continuously, since the attack can be blocked by an XMPP proxy.

In our solution, communication between an XMPP server and the proxy is not utilized using the XMPP protocol. Instead, a Hazelcast clustering protocol is used which operates

on top of the TCP. This mechanism offers natively proxy functionalities and prevents various attacks since both XMPP and TCP connections are terminated. Moreover, the solution enables Deep Packet Inspection (DPI) functionalities between the domains on the Hazelcast links. In the proposed architecture, proxies can also be used between security domains when they are acting as infromation exchange gateways (IEGs).

This article is organized as follows. We start with an overview of the XMPP and proxies in Section II. Our solution for an XMPP proxy for s2s connection is presented in Section III. We then discuss the future work of our solution in Section IV and conclude the article with recomendations in Section V.

II. XMPP and Proxies Overview

A. XMPP

The Extensible Messaging and Presence Protocol (XMPP) is a communications protocol for message-oriented middleware based on open Extensible Markup Language (XML), originally called Jabber [2]. The Internet Engineering Task Force (IETF) formed an XMPP working group in 2002 and this working group formed four RFCs (Request for Comments), which define the basic functionalities and protocols of the XMPP [6]–[9]. Furthermore, the XMPP Standards Foundation develops extensions to the protocol in the XEP series [10]. Currently there are approximately 170 extensions, XEPs, for the XMPP. Different XMPP server and client software exploit XEPs solely for their own interests.

The XMPP protocol supports presence information and contact list maintenance. Additionally, itl can be used in addition in publish-subscribe systems, signaling for VoIP, video streaming, file transfers, gaming and Internet of Things (IoT) applications.

The XMPP uses a client-server architecture – clients do not talk directly to one another, $i.e.$ the XMPP model is decentralized. There are different methods for both client-to-server (c2s) and server-to-server (s2s) connections. The connections can be either encrypted or unencrypted. All the messages are sent in an XML message structure, called stanza.

The basic idea of the XMPP is presented in Fig. 1. When a client *user1@example.com* wants to communicate with a client *user2@example.com*, it first establishes a client-to-server connection to communicate with a server hosting *user1*'s account information. Since the destination client, *user2*, resides in the same domain (*example.com*), the local server notifies via a c2s connection *user2* that *user1* wants to communicate with it.

When a client needs to communicate with a node residing in a different domain, such as a connection from *user1@example.com* to *jarvinen@example.fi*, the communication must be similarly first established between the source client and the hosting server. Since the server *example.com* does not have a direct connection to the destination client *jarvinen@example.fi*, it must first establish a server-to-server connection to the server hosting the destination node. Finally, the server $example.fi$ can communicate with the destination node and deliver the required transmissions to enable the communication between the source and destination nodes. The opposite end of the s2s connections is found with the help of DNS SRV records. The s2s connection is always directly connected to the opposite server not via relays or proxies [6], [7]. Depending on the client software and network topology, conventionally DNS is not required for the c2s connections as frequently as for the s2s connections.

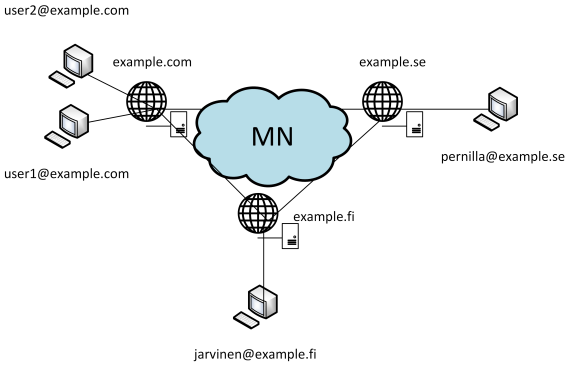

Fig. 1. The basic idea of XMPP.

B. Proxies

A proxy server is a method to split up a connection between a client and a server (see Fig. 2) or two clients. This kind of intermediators are used to hide information about private networks from adversaries. For example, we can secure the information about network topologies, IP addresses, connection pairs, physical locations [11] to increase network capacity by caching [12], or to scan content [13]. For each service, such as HTTP traffic, a separate proxy is required.

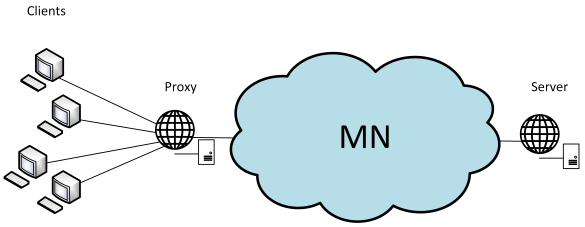

Fig. 2. An example of a proxy in use when it is located between clients and a server.

Conventionally, proxies are used in between private LANs (Local Area Networks) and the public Internet. The motivation may be to speed up an outbound Internet connection (caching in the proxy) or to hide the IP addresses of the client computers as presented in Fig. 2. Alternatively, clients do not have to know a complete route to the other end and vice versa. They both only see the proxy as a connection pair.

C. XMPP and Proxies

For c2s connections there are available open-source proxy implementations, for example, IMSpector. [14]. Several of those proxies act as connection managers by joining client connections to improve the scalability of the XMPP server as

illustrated in Fig. 3. Usually each XMPP server has a dedicated connection manager, for example, Openfire[1].

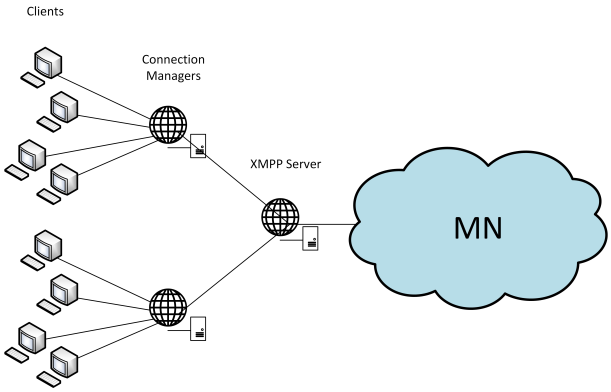

Fig. 3. The idea of XMPP c2s proxies.

Additionally, the Bidirectional-streams Over Synchronous HTTP (BOSH) technique [15] can be utilized in client proxies. BOSH is designed for asynchronous XMPP communication between a client and server using HTTP. This makes it possible for a client to reside, for example, on a web browser. In this case, the proxy is an HTTP proxy, not a pure XMPP. This scenario is illustrated in Fig. 4.

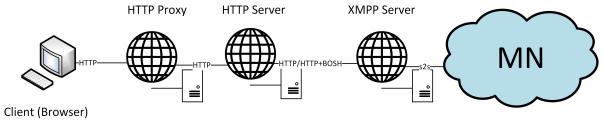

Fig. 4. The idea of the XMPP c2s proxy using BOSH.

Naturally, there are no proxies for s2s connections because the core RFCs of the XMPP protocol define that the s2s connections are directly connected to the opposite server where the other client resides [6], [7].

To enable interoperability of two IM protocols, a gateway mechanism between the protocols must be implemented. For example, a user using the XMPP can chat to a person, who is using the Internet Relay Chat (IRC) protocol when there is a gateway in between the networks. However, it is important to keep in mind that all IM networks do not support similar functionalities. For example, IRC does not support a user presence functionality. Otherwise, using an IRC node between two XMPP nodes requires an implementation of an XMPP proxy as described in Fig. 5. In addition, when the solution requires an extra component to a path, the service availability decreases.

Proxies are needed for s2s connections for the same reason as proxies are used for other services. For instance, proxies may be used to hide the internal structure of a network, hide connection pairs or perform content scanning.

[1]http://igniterealtime.org/projects/openfire/connection_manager.jsp

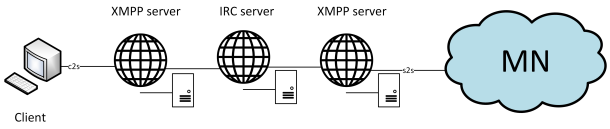

Fig. 5. The idea of using IRC between two XMPP nodes and, in this way, to produce an XMPP proxy.

III. Implementing XMPP s2s Proxy with Clustering Mechanism

As stated previously in Section II-C, there are no open-source XMPP proxies for server-to-server connections available on the markets. Thus, there are only two realistic options for building up a working solution of an XMPP s2s proxy:

1) To implement a XMPP server which acts like a proxy from scratch.
   - As Pros: This option is successful as long as there is enough resources and the platform is chosen correctly. The solution is fully independent.
   - As Cons: Time consumption. Source code requires constant updates and upgrades, when novel stanza models or XEPs are published.
2) To use an existing XMPP server and modify it to function like a proxy.
   - As Pros: When we are using a popular open-source XMPP software application which is updated constantly, we can be sure that the software is up-to-date, and follows latest RFCs and XEPs. All the SSL/TLS issues are solved.
   - As Cons: Finding a software solution where modifications can be added to work as a proxy. Solutions are not independent. Changes in new versions of original software may make our modifications useless or impossible to run.

After thorough discussions, we decided to reuse the source code of an existing XMPP server implementation. Hence, we could focus on the proxy functionalities and not the basic operational principles of the XMPP. As a basis for our implementation, we used Openfire XMPP servers (3.10.3)[2].

A. Architecture proposal

Our proposal for an XMPP server is based on the idea that the proxy functionality could utilize clustering protocols for s2s connections. In addition to the Openfire XMPP servers, we used a Hazelcast clustering plugin[3] in our implementation. The Openfire was chosen since it is supported by a viral and active community, continuously developing the software. In Openfire, new functionalities can be added by using plugins. Plugins are .jar packets with a certain internal structure.

Hazelcast is an open-source in-memory data grid which is based on Java. In a Hazelcast grid, data is distributed among the nodes of a computer cluster, allowing horizontal scaling

[2]http://www.igniterealtime.org/projects/openfire/
[3]https://www.igniterealtime.org/projects/openfire/plugins.jsp

in both speed and command execution. Hazelcast clustering is utilized in several open source products [16]. Hazelcast is implemented in Openfire using the plugin method.

One attractive benefit of utilizing the Hazelcast clustering is that in Hazelcast clustering the transmitted communication can not be observed as XMPP traffic, since the Stanza structure is removed from the packets prior to transmissions, and later re-constructed at the receiving end.

We built a topology as presented in Fig. 6 to study whether our clustering approach can be used to enable an XMPP proxy. There is no IP route from the MN to *xmpp2.example.se* or vice versa and no DNS record visible on the MN, only *xmpp1.example.se* has a connection to the Mission Network. Nodes *xmpp1* and *xmpp2* are in a Hazelcast cluster. Client A is connected only to the *xmpp2* node.

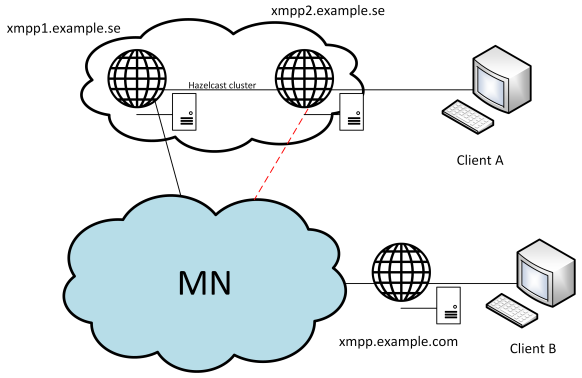

Fig. 6. The initial testing topology to find out whether a cluster mechanism would be a solution for enabling an XMPP proxy on s2s connections.

As a result we observed:

1) Client A cannot chat to Client B if there is not a connection (IP, DNS) between *xmpp2.example.se* and *xmpp.example.com* (dashed red line connection is down) but conversely Client B can chat to Client A.
2) Chatting in a Multi-User chat (MUC) room is working between clients A and B if the room resides at *xmpp2*.
3) Client A can chat to Client B if both clients already have an MUC session which is located at the *xmpp2.example.se* node.

It seems that some functions are working very well (Multi-User Chat) but private chat messaging between clients is functional only in one direction. The *xmpp2* node, although it is a cluster, tries to establish a new s2s connection from itself. If there already exists an old connection, for example, established by an MUC connection, we can use it, and private chatting in both directions works. During this initial test, there was an error *"Couldn't make an s2s connection, no route."* in the *xmpp2* log. The error may indicate that even in the cluster mode another node cannot initialize an s2s connection on behalf of a node where the fail happens. In other words, only the server serving the client can establish an s2s connection in Openfire cluster mode.

The problem seems to be the following: When Client A (in Fig. 6) tries to communicate with Client B, the server (xmpp2.example.se) tries to initiate the s2s connection. This initialization fails even if the server does not have a route to the opposite domain nor it receives a reply to the DNS query. However, in such cases that the proxy instance (xmpp1.example.se) has a connection to Client B, the XMPP connection between Client A and Client B can be established. The connection may be established, for instance, by an MUC connection or intialized by Client B.

A similar problem decreases the service availability even in normal Openfire Hazelcast clustering mode, since if a c2s connection is established with the node whose outgoing link is not working properly, connection to the outside is not possible.

B. Implementation

The correction for this problem requires modifications to Openfire source code. The required algorithm is presented in Algorithm 1: First, the algorithm must find out the node ID of the *xmpp1*. When the ID is found, the algorithm transmits all the s2s initialization queries to the node which handles them. Both nodes have unique IDs (Node ID) in the cluster mode, and with this ID any node can be commended. No modifications were made to the *xmpp1* server.

Algorithm 1: Algorithm for finding another node in a cluster and force it to initialize s2s connections. Static mapping parsing (lines 3-9) is an optional mechanism.

```
   Data: my own node ID OwnNodeID, chat domain
   Result: initialization of the s2s connection from the
           other node.
 1 cluster_entry_found=0;
 2 static_entry_found=0;
 3 while static_entry_found==0 && not at the end of the
     document do
 4     read static mapping file entrys
         (NextNodeID domains);
 5     if chat domain ≃ NextNodeID domain then
 6         NodeId = NextNodeID;
 7         static_entry_found=1;
 8     else
 9         static_entry_found=0;
10 while cluster_entry_found == 0 && not at the end of
     the list do
11     read next nodeID of the cluster, NextNodeID;
12     if OwnNodeID != NextNodeID then
13         NodeID=nextNodeID;
14         cluster_entry_found=1;
15     else
16         cluster_entry_found=0;
17 if cluster_entry_found || static_entry_found then
18     initialize s2s connection by using a node with
         NodeID;
```

C. Result and Testing

As a result, we were able to succesfully get two Openfire XMPP servers connected with a Hazelcast clustering link. With such a connection, they function as a single Openfire server. Hence, a Hazelcast cluster works in parallel as presented in Figure 7. Openfire needs a common external user database which is also included in the figure.

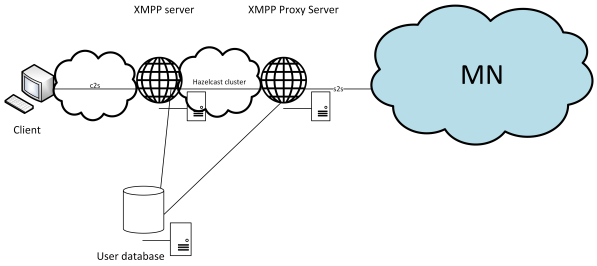

Fig. 7. The structure of a working solution of an XMPP s2s proxy in Openfire. The IP networks can also include protecting mechanisms, for example, firewalls, deep packet inspection (DPI) functionalities, intrusion prevention systems (IPSs), or data loss prevention (DLP) systems.

After the modifications, Instant Messaging (IM) which was a problem in Section III-A, also began to work properly. IM is working in the both directions: Client A $\longrightarrow$ Client B and Client B $\longrightarrow$ Client A (See Figure 6) without any "initialization" by MUC room traffic as was the case initially.

Our solution also improves the route availability in Hazelcast clustering, and hence also the service availability as dedmonstrated in Fig. 6 can be improved internally as well.

In Figure 8 is illustrated the flow diagram of the transmission of one message in our XMPP proxy implementation. After the XMPP server receives a message from a client, it simultaneously transmits an acknowledgement of the received message to the client and informs the XMPP proxy of an upcoming message. Then the XMPP proxy establishes a server-to-server connection to the XMPP server of the required domain, and delivers a message to the server. Finally, the server is again reponsible for transmitting the message to the correct client. Each of these flows are utilizing TCP.

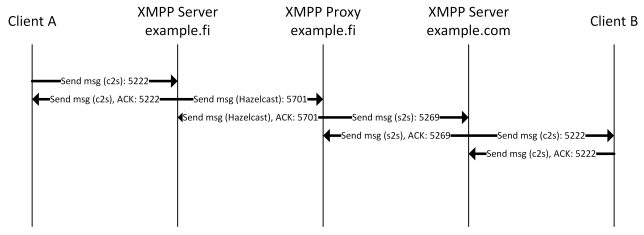

Fig. 8. The flowchart of our XMPP proxy implementation.

We have also studied the impact of our proposal from the perspective of connection delay. First, we tested the normal situation with two domains communicating without an XMMP proxy approach. The test scenario is illustrated in Fig. 9. The second scenario is presented in Fig. 10. In the second test scenario the XMPP proxy server was utilized. Both scenarios were tested under two cases: In the first case, messages were transmitted frequently, and the s2s connection was not disconnected between message transmissions. In the latter case, the s2s connection was disconnected between two consecutive messages, and hence the connection was initialized for each message transmission. The delay was measured from the initializing TCP segment (SYN) of the TCP connection from Client A to the acknowledging (SYN,ACK) TCP segment from the MN. The sample count for the test scenarios was between 270 and 2080. The transmitted message was a short 5 byte message from Client A to Client B.

Fig. 9. Single node test scenario.

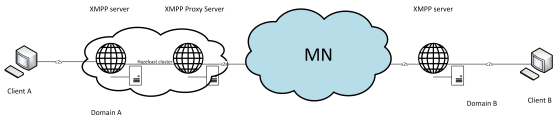

Fig. 10. Test scenario with proxy.

The results of this study are presented in Fig. 11. The results demonstrate that our proposal does not impact significantly on the anticipated delay. For instance, if the interval between consecutive messages were short, the expected delay, if the XMPP proxy was not used, was approximately 60 ms. Similarly, when the proxy was utilized, the expected delay was approximately 70 ms.

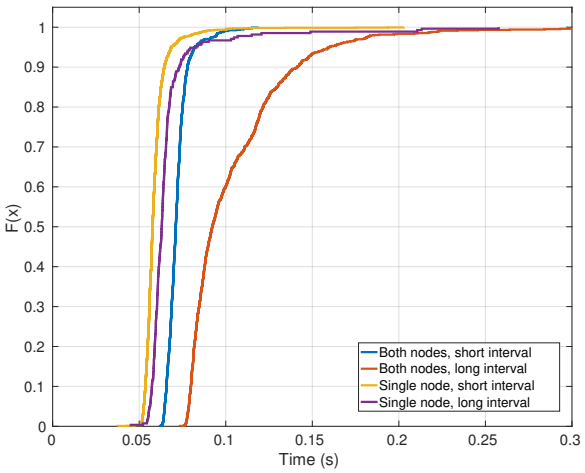

Fig. 11. The cumulative distribution function of delays.

D. Limitations

In this implementation we assume that all the other XMPP domains reside behind a proxy node; all the server-to-server traffic is directed to the proxy node. However, when needed, such limitations can be dissolved easily by defining a static

mapping/routing mechanism. The static mapping file includes entries to combine NodeIDs and chat domain names. The algorithm goes through all chat domains presented in the file, and compares them to the domain it wants to communicate with. When the equivalent entry is found, also the required NodeID is discovered. The functionality how the approach utilizes the static mapping is presented in Algorithm 1 lines 3-9.

E. Pros and Cons of Implementations

The benefit of our implementation is that, the approach functions like a conventional proxy. Hence, a TCP connection is terminated at the servers. Furthermore, since a cluster link is used, only the content of messages is forwarded between the nodes, not the XMPP stanza structure. This means, that for example, attacks targeting the XMPP stanza structure are blocked automatically. At the cluster link we are able to implement DPI functionalities, for example, virus scanning and a sanity check, even TLS is used on s2s connections and/or c2s connections. The cluster link also enables the proxy to act as an IEG to enable the interoperability between different domains at various security levels. On the other hand, however, it enables filtering unauthorized communications with Data Loss Prevention (DLP) mechanism and implementing release control between security domains.

The only drawback of this architecture is the dependency of the software used in the XMPP server – the proxy server software version has to be the same across all nodes. In such a case, one single vulnerability can give an attacker complete access to an internal network.

IV. Future Work

For now the approach focuses on single servers and they are not redundant in any way. However, we think that clustering is not a problem and it is possible to define separately proxies and conventional servers in the source code, as presented in Fig. 12. The functionalities of Hazelcast take care of the rest. Nevertheless, this should still be built up and tested in order to be verified.

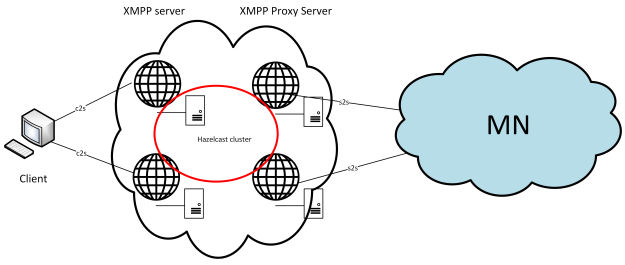

Fig. 12. The structure of a backuped XMPP proxy with a Hazelcast cluster method.

V. Conclusion

In this paper we proposed an architecture of an XMPP security proxy for s2s connections to improve security and prevent malicious attacks toward an XMPP server.

We have implemented the presented XMPP proxy architecture by exploiting a Hazelcast clustering plugin for an Openfire XMPP server, and by modifying source code of Openfire. Our solution enables Deep Packet Inspection functionalities between domains on a Hazelcast clustering link. Since messages are transferred between nodes without any XMPP stanza structure, our solution could also act as an IEG. By using our architecture there is no longer any direct XMPP s2s connection between local and remote XMPP servers. In addition to DPI functionalities, the approach enables attaching numerous nodes in the XMPP clusters. This enables load sharing, and complicates the implementation of denial-of-service attacks.

Our implementation is only suited for system where Openfire is used as an XMPP server with a Hazelcast clustering plugin. Generally, we cannot state that there is an option to produce an XMPP proxy with the help of clustering by using other software even though that may be capable of clustering.

As a by-product we have also sketched an improvement to increase the availability of a clustered Openfire XMPP server.

References

[1] J. Oikarinen and D. Reed. Internet Relay Chat Protocol. Legacy RFC 1459, March 2013.

[2] The XMPP Blog: What is XMPP? https://xmpp.org/2007/10/what-is-xmpp/. [Online; accessed 27-January-2016].

[3] Robert Lass, Joe Macker, David Millar, C Regl William, and Taylor Iani. XO: XMPP overlay service for distributed chat. In *Military Communications Conference, MILCOM 2010*, pages 1116–1121. IEEE, 2010.

[4] Robert N Lass, Joe Macker, David Millar, and Ian J Taylor. Gump: adapting client/server messaging protocols into peer-to-peer serverless environments. In *Proceedings of the 2nd workshop on Bio-inspired algorithms for distributed systems*, pages 39–46. ACM, 2010.

[5] M. Skjegstad, K. Lund, E. Skjervold, and F. T. Johnsen. Distributed chat in dynamic networks. In *Military Communications Conference, MILCOM 2011*, pages 1651–1657, Nov 2011.

[6] P. Saint-Andre. Extensible Messaging and Presence Protocol (XMPP): Core. IETF RFC 3920, October 2015.

[7] P. Saint-Andre. Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence. IETF RFC 3921, October 2015.

[8] P. Saint-Andre. Mapping the Extensible Messaging and Presence Protocol (XMPP) to Common Presence and Instant Messaging (CPIM). IETF RFC 3922, October 2015.

[9] P. Saint-Andre. End-to-End Signing and Object Encryption for the Extensible Messaging and Presence Protocol (XMPP). IETF RFC 3923, October 2015.

[10] XMPP - xmpp.org. [Online; accessed 06 February 2016].

[11] V. Kambhampati, C. Papadopolous, and D. Massey. Epiphany: A location hiding architecture for protecting critical services from DDoS attacks. In *2012 42nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 1–12, June 2012.

[12] Marc Necker, Michael Scharf, and Andreas Weber. Performance of tcp and http proxies in umts networks. In *11th European Wireless Conference 2005 - Next Generation Wireless and Mobile Communications and Services (European Wireless)*, pages 1–7, April 2005.

[13] Dwen-Ren Tsai, A.Y. Chang, Sheng-Chieh Chung, and You Sheng Li. A proxy-based real-time protection mechanism for social networking sites. In *2010 IEEE International Carnahan Conference on Security Technology (ICCST)*, pages 30–34, Oct 2010.

[14] IMSpector. http://www.imspector.org/. [Online; accessed 03 April 2016].

[15] I. Paterson, D. Smith, P. Saint-Andre, J. Moffitt, L. Stout, and W. Tilanus. XEP-0124: Bidirectional-streams Over Synchronous HTTP (BOSH), April 2014.

[16] Hazelcast The Operational In-Memory Computing Platform. https://hazelcast.com/. [Online; accessed 12 February 2016].