DE GRUYTER OPEN

Int. J. Appl. Math. Comput. Sci., 2016, Vol. 26, No. 2, 479–493
DOI: 10.1515/amcs-2016-0034

amcs

A BRANCH HASH FUNCTION AS A METHOD OF MESSAGE SYNCHRONIZATION IN ANONYMOUS P2P CONVERSATIONS

ANNA KOBUSIŃSKA [a,*], JERZY BRZEZIŃSKI [a], MICHAŁ BOROŃ [a], ŁUKASZ INATLEWSKI [a], MICHAŁ JABCZYŃSKI [a], MATEUSZ MACIEJEWSKI [a]

[a] Institute of Computing Science
Poznań University of Technology, ul. Piotrowo 2, 61-131 Poznań, Poland
e-mail: {akobusinska,jbrzezinski,mboron}@cs.put.poznan.pl,
{linatlewski,mjabczynski,mmaciejewski}@cs.put.poznan.pl

Currently existing solutions rarely protect message integrity, authenticity and user anonymity without burdening the user with details of key management. To address this problem, we present Aldeon—a protocol for anonymous group conversations in a peer-to-peer system. The efficiency of Aldeon is based on a novel tree synchronization algorithm, which is proposed and discussed in this paper. By using this algorithm, a significant reduction in the number of exchanged messages is achieved. In the paper, the formal definition of the proposed hash branch function and the proof of its efficiency are presented.

Keywords: peer-to-peer, synchronization, conversations, anonymity.

1. Introduction

The enormous growth of the Internet has initiated various new trends. It has become common to use forums, instant messaging services, social networks and other tools enabling group communication (e.g., Facebook, Reddit, Twitter). Consequently, exchanging information, finding people of similar opinions, and organizing meetings and social movements have become common (Moore and Zuev, 2005; Xiao *et al.*, 2007; Xie and Wang, 2012; Baruah, 2012). However, along with the continuous growth in the popularity of group communication tools and services, their drawbacks also become more apparent (Sakarindr and Ansari, 2010; Völker *et al.*, 2011).

Many currently existing communication tools and services rely on dedicated servers (Zhang *et al.*, 2010; Miller, 2014). Such servers can be set up, maintained and accessed relatively easily. But the reliability of message servers depends on the proprietary infrastructure, and thus constitutes a single point of failure. Moreover, participants are required to trust the central service, being a third party. This means that the anonymity and integrity of messages depends on the good will and professionalism of the service administrators. It can be easily seen that this may lead to problems related to privacy and censorship. Finally, in the most common cases, connections between communication participants and message servers are direct, which makes them vulnerable to spying.

The problems mentioned above may be addressed by employing a peer-to-peer (P2P) system model (Schollmeier, 2001; Schoder and Fischbach, 2003; Lv *et al.*, 2012; Aditya *et al.*, 2014). In such a model, nodes (also called peers) interact directly with each other in order to utilize shared resources. Resources serve various purposes—most commonly they are used to provide service or content, e.g., file uploading and downloading, streaming multimedia or instant messaging. The emerging peer-to-peer concepts provide new possibilities—their architecture makes the system independent of any supervisor, thus eliminating the single point of failure. Furthermore, decentralization makes the system more scalable, and reduces the risk of conversation censorship or control. In addition, the availability of services increases with the number of users.

With these concerns in mind, in this paper we propose Aldeon—a peer-to-peer protocol that enables anonymous group communication. In Aldeon, nodes

*Corresponding author

participate in conversations by publishing and exchanging posts organized into topics. A node collects and stores posts related to the chosen topic, and makes them available for other nodes to download. Topics form tree structures—each post can have any number of response posts. As can be seen, e.g., in Reddit (Weninger, 2014), trees that represent human conversations are massive and almost never balanced. Consequently, an efficient synchronization mechanism that enables fast exchange of posts between nodes is necessary.

Although the idea of exchanging posts and ensuring the eventual consistency of shared data structures seems to be straightforward, in fact protocols providing such a mechanism face several problems. In P2P systems, epidemic protocols are commonly applied to exchange information on a large scale. The eventual consistency of a shared topic tree structure can thus be achieved by epidemic broadcasting of either individual posts or of the entire state of a topic data structure. Although it may seem that broadcasting posts is better (due to a smaller data transfer overhead), this is not always the case. It must be ensured that the post recipient already has its causal dependencies (ancestor posts). A naive approach would be to gossip the whole tree structure periodically, but that would introduce a large overhead. This solution can be improved by exchanging only the missing posts. It is difficult to determine which posts should be downloaded because of constant changes in the conversation held by the nodes. These changes may be resulting from adding new posts or deleting those already possessed by others (Gilbert and Lynch, 2002).

To address this problem, we present our approach to synchronizing tree structures by introducing a gossip algorithm based on a branch hash function (BHF). The proposed solution reduces synchronization overhead and enables fast exchange and synchronization of posts. In the following sections, we describe this function's properties and show its efficiency.

The paper is structured as follows. The system model and basic assumptions are presented in Section 2. Section 3 describes the general idea of the branch hash function, and discusses its advantages and limitations. Next, Section 4 presents the Aldeon protocol, which uses the introduced branch hash function in order to synchronize the information on exchanged posts that make up the conversation. The discussion on Aldeon protocol security considerations is conducted in Section 5. Related work is characterized in Section 7. Finally, Section 8 concludes the paper.

2. System model and basic assumptions

Throughout this paper, a peer-to-peer (P2P) system model (Schollmeier, 2001; Schoder and Fischbach, 2003) is considered. The system consists of nodes similar in role, function and capabilities, which act simultaneously as clients and servers. Nodes interact with each other and exchange posts to take part in conversations.

All conversations in the system model considered are organized into *topics*. To interact with others, a user (author of posts) may create topics, download posts written by other users and respond to them. The most often used form of representing conversations in the Internet is a list of posts, ordered by date. While useful in small conversations, it is not comfortable for application when the number of participants grows, mainly because it leads to connecting independent threads into one stream of messages. As a result, in the paper we assume that posts which make up the conversation, form a *tree* structure (each conversation forms a separate tree). The root post of such a tree is called a conversation topic. Every other post is treated as a response to another post, called its parent. Since posts in a tree structure point to their parents, each response causally depends on its parent. The posts considered in this paper are represented by a quadruple of the following variables: the parent post identifier, the author public key, the post content and the signature (generated using the author private key). A distributed hash table (DHT) (Dabek *et al.*, 2004) is used to organize nodes according to their topic preferences. As a result, posts are delivered epidemically only to those nodes that seek them.

We assume that nodes can be malicious, and as such they may generate corrupted messages (either intentionally or because of an incorrect implementation) (Laprie *et al.*, 1992). Consequently, nodes cannot be trusted, and the criteria for asserting whether or not a given post is correct are needed. In the case of a protocol in which each post has its author, a natural choice of the post integrity validation method is signature correctness criteria. We provide this mechanism using the RSA algorithm (Rivest *et al.*, 1978; ENISA, 2012). Each *user* (author of posts) in the protocol possesses a pair of asymmetric keys, from now on referred to as the *identity*. The term *user* refers to post author and is not directly related to any node, though in the most common scenario each user will control its own node. The public key acts as the user identifier, while the private key is used to sign posts. Thus, nodes are able to identify the author of a post by combining the public key of the given identity with the post.

As a consequence of using the RSA algorithm, all public keys in the system considered are unique. However, it is not possible to determine which identity is assigned to which node. This anonymity in the system is based on the fact that there is no mapping between nodes and user identities, and this is known as *plausible deniability* (Boyd *et al.*, 2005; Saxena *et al.*, 2014).

3. Posts synchronization

In Aldeon, each node should eventually receive all posts associated with a given conversation. Each node uses a distributed hash table to find a subset S of conversation participants. Afterwards, an instance of the synchronization algorithm is ran for each member of S. The purpose of the synchronization algorithm is to efficiently download missing posts from another node. The exchanged posts are selected based on the analysis and comparison of different branches of the conversation tree. The algorithm traverses the topic tree from the root to leaves, processing each branch (possibly in parallel). As a result, all posts that are not stored locally are gradually detected and downloaded. After the synchronization, nodes periodically send updates about the posts that have arrived since the last update. This way nodes can stay synchronized for as long as necessary.

The proposed algorithm tackles the problem of synchronizing unbalanced trees. It may happen that the tree is formed as a long list, and in that case synchronization would require many consequent queries and responses. Network latency may make this process prohibitively expensive in terms of time. On the other hand, sending all known posts in advance could involve transferring large amounts of unnecessary data. To solve this problem, we introduced a *branch hash function* that allows identifying the differences in the trees stored by nodes. The proposed function significantly reduces the amount of exchanged data and the number of requests necessary to complete the synchronization process.

3.1. Branch hash function. Let us denote by $\otimes$ a XOR function. XOR forms an abelian group over $\mathbb{Z}_n$. Furthermore, every value is its own inverse under the XOR operation:

$$a \otimes 0 = a, \tag{1}$$

$$a \otimes b = b \otimes a, \tag{2}$$

$$a \otimes (b \otimes c) = (a \otimes b) \otimes c, \tag{3}$$

$$a \otimes a = 0. \tag{4}$$

We will write a XOR sum of multiple values as

$$\bigotimes_{e \in E} e \equiv e_1 \otimes e_2 \otimes ... \otimes e_n, \quad e_i \in E. \tag{5}$$

Each post i is identified by I_i, called the post identifier. The post identifier is an integer in the $\mathbb{Z}_n$ modulo group. Each post has a parent post identifier, denoted by $P(I)$. For the root post (topic post), the parent post identifier equals 0:

$$Root(I_r) \equiv P(I_r) = 0. \tag{6}$$

A *children set* is defined as follows:

$$Ch(I_n) \equiv \{I_c : P(I_c) = I_n\}. \tag{7}$$

We define a *descendant set* of a post I_n as a set of all posts that form a branch descending from the post I_n:

$$De(I_n) \equiv \{I_d : I_d \in Ch(I_n) \vee P(I_d) \in De(I_n)\}. \tag{8}$$

Let us now define a *branch hash function*, denoted by $S(I_b)$:

$$S(I_b) \equiv I_b \otimes \bigotimes_{I_c \in Ch(I_b)} S(I_c). \tag{9}$$

From the commutativity (2) and the associativity (3), the branch hash function $S(I_b)$ can be rewritten in the following way:

$$S(I_b) = I_b \otimes \bigotimes_{I_d \in De(I_b)} I_d. \tag{10}$$

If the post has no children, the value of a branch hash function $S(I_b)$ is equal to its post identifier:

$$(Ch(I_b) = \varnothing) \Rightarrow S(I_b) = I_b. \tag{11}$$

Figure 1 shows an example of a tree and its branch hash function values.

3.2. Properties of the branch hash function. In this section we prove that the proposed branch hash function selects only the differences in trees of posts stored in the nodes. As a result, posts that have been already possessed by nodes are not exchanged.

Let W_n denote a set of identifiers of posts stored by a node n. A set W_n forms a single tree, as it contains exactly one root post ((12), (13)). It is assumed that a set W_n is consistent, i.e., for each post in the set, its parent is also in the set (14):

$$\exists I_a \in W_n : Root(I_a). \tag{12}$$

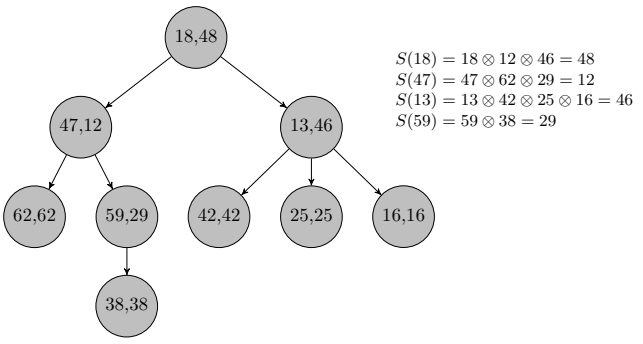


Fig. 1. Example of correctly calculated branch hash function values.

$$\nexists I_a, I_b \in W_n : Root(I_a) \wedge Root(I_b). \quad (13)$$

$$(I_q \in W_n \wedge \neg Root(I_q)) \Rightarrow P(I_q) \in W_n. \quad (14)$$

Let us consider nodes a and b. Nodes store sets of post identifiers W_a, W_b, including the root post identifier I_r. Further, let us denote by S_a a value of a branch hash function, calculated with the use of posts stored in the set W_a. S_b is defined analogously for the node b. We define the branch hash value of an unknown post to equal 0:

$$(I \notin W_n) \Rightarrow S(I) = 0. \quad (15)$$

If both the nodes possess the same posts, the values of the branch hash function for nodes a and b are equal:

$$(W_a = W_b) \Rightarrow S_a(I_r) = S_b(I_r). \quad (16)$$

In turn, if the post sets W_a, W_b differ from each other, there exist messages stored only by one of these nodes. Let the symbol $\bigoplus$ denote a symmetric difference of sets:

$$A \bigoplus B = \{x : x \in A \setminus B \vee x \in B \setminus A\}. \quad (17)$$

Based on the above properties, we show that for the branch hash function the following theorem holds.

Theorem 1. *The branch hash function makes it possible to find a XOR sum of posts constituting a difference between the trees:*

$$S_a(I_r) \otimes S_b(I_r) = \bigotimes_{I_x \in W_a \bigoplus W_b} I_x. \quad (18)$$

Proof. On account of (12)–(14), we have

$$De_a(I_r) \cup \{I_r\} = W_a. \quad (19)$$

This means that from (10) we get

$$S_a(I_r) = \bigotimes_{I_n \in W_a} I_n. \quad (20)$$

Thus, the left-hand side of (18) can be rewritten as follows:

$$\left(\bigotimes_{I_i \in W_a} I_i\right) \otimes \left(\bigotimes_{I_i \in W_b} I_i\right). \quad (21)$$

This can be further expanded to

$$\left(\bigotimes_{I_i \in W_a \setminus W_b} I_i\right) \otimes \left(\bigotimes_{I_i \in W_a \cap W_b} I_i\right) \quad (22)$$

$$\otimes \left(\bigotimes_{I_i \in W_b \setminus W_a} I_i\right) \otimes \left(\bigotimes_{I_i \in W_a \cap W_b} I_i\right).$$

Using Eqn. (4), this can be reduced to

$$\left(\bigotimes_{I_i \in W_a \setminus W_b} I_i\right) \otimes . \left(\bigotimes_{I_i \in W_b \setminus W_a} I_i\right). \quad (23)$$

This leads to the right-hand side of (18) of which ends the proof:

$$\bigotimes_{I_x \in W_a \bigoplus W_b} I_x. \quad (24)$$

■

3.3. Examples of and difficulties in using the branch hash function. Storing precomputed values of the branch hash function allows fast identification of differences in sets W_n between nodes. As shown in Eqn. (16), for the same sets W_n the sum of differences equals 0. When one of the sets contains another, the sum of differences is the following:

$$W_b \subseteq W_a \Rightarrow S(W_b) \otimes S(W_a) = \bigotimes_{I_x \in W_a \setminus W_b} I_x. \quad (25)$$

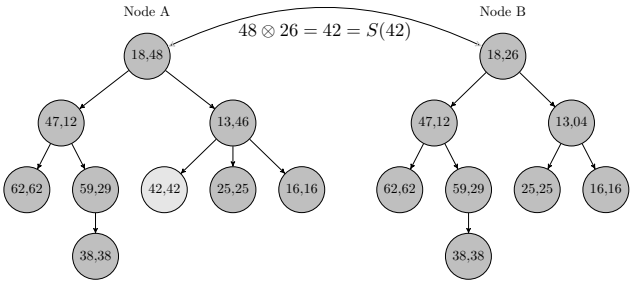


Fig. 2. Node A has one additional message in comparison to node B.

By adding the values of the branch hash function, the nodes get insight into what value the missing elements sum up to. If the difference forms a single branch, the returned value allows determining which branch has to be sent in order to synchronize trees. This scenario is shown in Fig. 2, where the only difference is a single node, and in

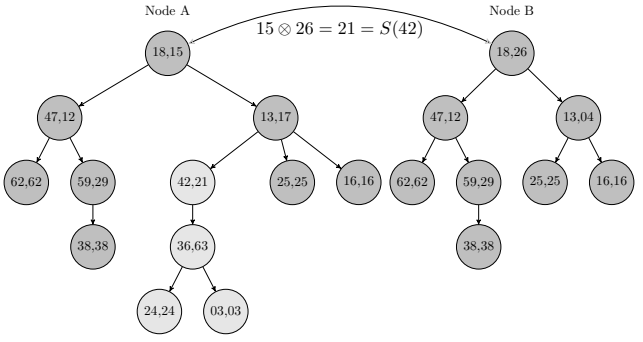


Fig. 3. Node A has one additional branch in comparison to node B.

Fig. 3, where the difference forms a branch of messages. Differences in multiple places remain a problem, as it is impossible to decompose the sum into its summands. As shown in Fig. 4, a branch hash function indicates the existence of differences, but not their location. It should be noted that the uniqueness of posts does not imply the uniqueness of values returned by the branch hash function. This leads to two important conclusions:

1. The equality of values returned by function $S(I_n)$ for two trees does not imply the equality of these trees. In other words, the implication (16) works only in one way.
2. Finding a post I_n for which $S(I_n) = S_a(I_r) \otimes S_b(I_r)$ does not mean that this particular post is missing in the other node.

The probability of the occurrence of events described above is related to the size of the modulus in group $\langle \bigotimes, \mathbb{Z}_n \rangle$. Let n be the number of bits needed to store any message identifier:

$$P\Big(S_a(I_r) = S_b(I_r) \wedge W_a \neq W_b\Big) = \frac{\max(\overline{W_a}, \overline{W_b})}{2^n}. \tag{26}$$

The impact of this probability on the stability of the synchronization protocol is presented in Section 5.

4. Synchronization protocol based on the branch hash function

In this section we present a synchronization protocol that allows nodes to effectively exchange missing posts on a given topic. The protocol employs recursive message tree comparison, and uses the branch hash function, introduced in Section 3.1, to minimize the number of exchanged messages. First, the general idea of synchronization based on the branch hash function is described. Afterwards, the proposed synchronization protocol is presented.

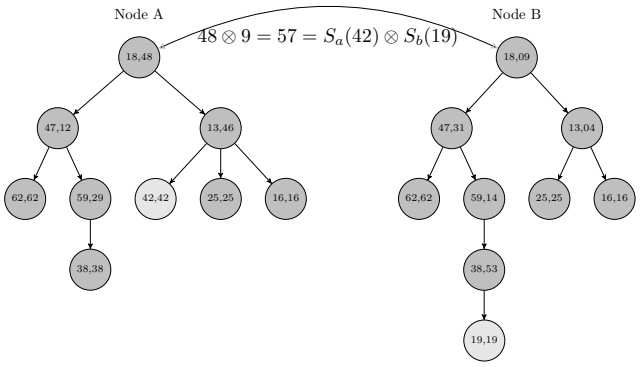


Fig. 4. Both nodes have posts, which are unknown to the other party.

4.1. General idea of synchronization. Each node, in order to synchronize its knowledge on the given branch containing the post with the information held by other nodes, performs the synchronization procedure shown in the form of a block diagram in Fig. 5.

The presented protocol is one-way—each peer must perform the algorithm separately to identify and download missing posts. The procedure, described by the block diagram in Fig. 5, can occur on the same connection and is safe to run concurrently in both ways.

Let us denote by N_A the local node (performing the procedure), and by N_B the remote node (answering to requests). According to Fig. 5, the node N_A performs the following steps: it calculates a local hash value $S_A(I)$ of the branch I and sends a request for a branch hash value comparison to the remote node. The request contains the identifier of the post I and the obtained hash value $S_A(I)$.

As a result of obtaining a request for branch comparison, the remote node N_B responds in one of three ways: if the values of the branch hash function $S_A(I)$ and $S_B(I)$ are equal, then the procedure terminates because the nodes are synchronized. Otherwise, if the non-zero difference branch hash value corresponds to one of the branch hash values in N_B, a $Suggest$ response indicating this branch is sent back. Upon receiving the $Suggest$ response, the node N_A downloads the suggested message together with its descendants and also terminates. Finally, if a $Children$ response with the children list is returned, the synchronization procedure is recursively called for each child.

In the latter case, the following optimizations can

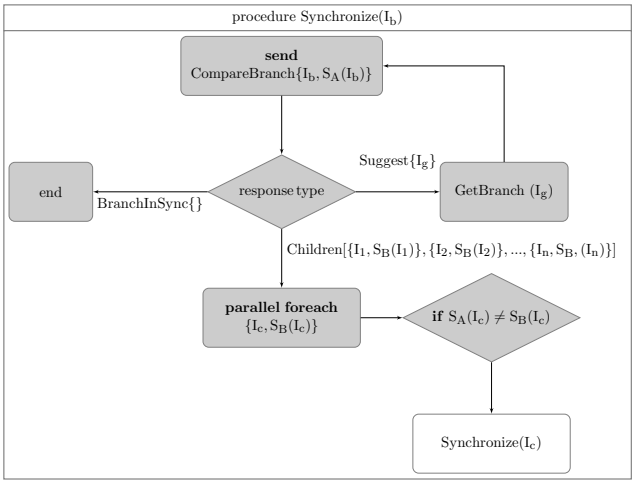


Fig. 5. Simplified synchronization process.

be applied: first, the analysis of the message list can be run in parallel. Furthermore, since a reply containing missing messages has the form of a list of pairs $(I, S(I))$, the node is able to compare hash values, and stop the synchronization process if the sub-branches are equal. It is also able to deduce if the tree differences are located in local memory. This way the messages can be sent directly to the server, effectively turning synchronization into a two-way process.

Despite the speedup obtained by using the branch hash function, the synchronization process can still be time-consuming because of request transmission time. Therefore, to maintain a coherent state after the initial synchronization, each node can maintain a *logical clock* and increment it for each new post stored locally. By periodically asking for posts that were created after the last observed clock value, nodes can keep up with the changes and stay consistent.

4.2. Synchronization protocol. The proposed protocol is executed by each node in the system.

Nodes use a DHT mechanism to find other nodes interested in following the same topics, thus enabling efficient post propagation using the epidemic approach. The DHT implementation is not in the scope of this paper and thus, for the clarity of presentation, here we omit its implementation details.

In the proposed protocol we use types and data structures presented in Algorithm 1.

Algorithm 1. Types and data structures.

define ID $uint256$
define $Post$ $\{id, parent : ID, content\}$
define $GetBranch$ $\{id : ID\}$
define $CompareBranch$ $\{id, hash : ID, f : bool\}$
define $BranchFound$ $\{posts : Post[]\}$
define $Suggest$ $\{id, parent : ID\}$
define $Children$ $[\{id, hash : ID\}, ...]$
define $BranchInSync$ $\{\}$
define $BranchNotFound$ $\{\}$

The ID type represents a post identifier—in the case of our protocol, a 256-bit vector. In the minimal case, each post contains its identifier, its parent identifier and content. The cryptographic aspects are not included in the algorithm description, as they are independent of the stated problem. The only guarantee required by the algorithm is that posts have distinct, uniformly random identifiers.

The algorithm begins when the initiator node (N_A) obtains an identifier I_b of the branch it wants to synchronize. Algorithm 2 is executed with the I_b identifier and the *false* flag. The node checks if the post is already stored (Algorithm 2, line 1), and downloads it from the remote node I_B if necessary (Algorithm 2, lines 2–4); Algorithm 4. Then, the branch hash value $S_A(I_B)$ is computed (or fetched from storage, if it was precomputed). The identifier, branch hash and flag are sent to the remote node in a $CompareBranch$ request (Algorithm 2, line 9). The remote node handles the request (Algorithm 3) by first checking if the post is known (Algorithm 3, line 1). If not, the $BranchNotFound$ is returned and the synchronization process terminates (Algorithm 3, line 2). Otherwise, in the case of the known post, the remote node calculates its own branch hash $S_B(I_b)$ (Algorithm 3, line 4) and compares it with the received value (Algorithm 3, line 5). When the values are the same, there is a high probability that the branches are equal (Eqn. (26)). In such a case the $BranchInSync$ message is returned to inform the initiator node that there are no more posts to download (Algorithm 3, line 8). In turn, if the values differ, the node checks whether there are any known branches that have a matching branch hash value (Algorithm 3, line 10). Consequently, these values can be precomputed.

Each time a match is found, the node suggests downloading the particular branch by sending the $Suggest$ response (Algorithm 2, line 10). After obtaining a response, the node N_A verifies whether the post is in fact not known and the dependencies are met (the parent post should be in currently synchronized branch) (Algorithm 2, line 11). It is only when the above the conditions are met that the node synchronizes the suggested branch and then retries synchronizing the previous one (Algorithm 2, lines 12–13). After that, the procedure needs to be carried out again, as the suggested branch may be just one of the missing branches. Consequently, synchronizing again with the node N_A ensures that eventually all branches are in sync. If the suggested branch does not meet the conditions, the synchronization procedure is repeated with the flag set to *true* (Algorithm 2, line 14). This way the next synchronization attempt will proceed without any suggestions, and the *Children* response will be returned.

When receiving the $Children$ response (Algorithm 2, line 17), the node N_A tries to synchronize the child branches. These subbranches can be processed in parallel (Algorithm 2, lines 18–22). When the synchronization procedure terminates, the initiator node N_A has all the posts known by node N_B.

5. Security considerations for communication in the Aldeon protocol

In the Aldeon protocol, some of the participating nodes are assumed to behave incorrectly. In most cases security concerns for peer-to-peer computing (Sit and Morris, 2002; Damiani *et al.*, 2004) tend to focus on the attacks on routing and lookup protocols that use distributed hash tables or on attacks related to the storage and

Algorithm 2. Synchronization procedure.

Procedure Synchronize $(I_b : ID, f : bool)$
1: **if** $I_b \notin W_A$ **then**
2: $rsp \leftarrow$ **send** $GetBranch\{I_b\}$
3: **if** rsp **is** $BranchFound\{posts\}$ **then**
4: $W_A \leftarrow W_A \cup \{posts\}$
5: **end if**
6: `// No more posts to download`
7: **return**
8: **end if**
9: $rsp \leftarrow$ **send** $CompareBranch\{I_b, S_A(I_b), f\}$
10: **if** rsp **is** $Suggest\{I_g, P(I_g)\}$ **then**
11: **if** $I_g \notin W_A$ **and** $P(I_g) \in De_A(I_b)$ **then**
12: $Synchronize(I_g, false)$
13: $Synchronize(I_b, false)$
14: **else**
15: $Synchronize(I_b, true)$
16: **end if**
17: **else if** rsp **is** $Children[\{I_1, S_B(I_1)\}, ...]$ **then**
18: **for all** $\{I_c, S_B(I_c)\} \in rsp$ **do**
19: **if** $S_A(I_c) \bigotimes S_B(I_c) \neq 0$ **then**
20: $Synchronize(I_c, false)$
21: **end if**
22: **end for**
23: **else**
24: `// BranchInSync or BranchNotFound`
25: **end if**

Algorithm 3. $CompareBranch$ request handler.

Handle $CompareBranch(I_b, S_A(I_b) : ID, f : bool)$
1: **if** $I_b \notin W_B$ **then**
2: **return** $BranchNotFound$
3: **end if**
4: $diff \leftarrow S_A(I_b) \bigotimes S_B(I_b)$
5: **if** $\neg f$ **and** $\exists I_g : S_B(I_g) = diff$ **then**
6: **return** $Suggest\{I_g, P(I_g)\}$
7: **else if** $diff = 0$ **then**
8: **return** $BranchInSync$
9: **else**
10: **return** $Children[\{\{I_c, S_B(I_c)\} : I_c \in Ch(I_b)\}]$
11: **end if**

Algorithm 4. $GetBranch$ request handler.

Handle $GetBranch(I_p : ID)$
1: **if** $\exists post \in W_B : post.id = I_p$ **then**
2: **return** $BranchFound\{\{post\} \cup De(I_p)\}$
3: **else**
4: **return** $BranchNotFound$
5: **end if**

retrieval of resources. In this section, the most common attacks associated with the exchanging of messages are presented. In particular, the problems occurring when unsolicited messages are sent or when some messages are purposely omitted while communicating with other nodes are discussed. Examples of how these attacks can occur in the context of the Aldeon protocol will be shown, along with some methods for detecting and preventing these problems.

5.1. Unsolicited posts. Due to the equal rights of all participants of the conversation, which is particularly evident in the case of communication in a P2P network, each participant is able to send a message to all other nodes participating in the conversation.

Preventing unsolicited messages in P2P systems differs from traditional solutions used in distributed systems. Blocking messages issued by a given author is not effective, because the message sender can easily change his/her identity. Given the impossibility of determining who is the owner of the identity, it is difficult to resist such an attack. One of the methods to counter the unsolicited messages used in distributed systems is to block the specific IP address from which the unsolicited message was received. In the context of P2P systems such a solution would not bring the desired effect, because there is no way of determining whether the sender node is the author of the message, or if it has received the message from another node and simply passed it on. Blocking certain nodes is also not viable as unsolicited messages can still propagate to other nodes and reach them through a different network path.

The problem with receiving unsolicited messages must be resolved in the context of the Aldeon protocol. In the protocol, nodes are not required to know the name of a post recipient in order to publish a post. Given the public availability of conversation identifiers, and the possibility of obtaining such an identifier from a request issued by another node, it can be assumed that the prevalence of such unsolicited messages is a real threat, affecting the accuracy and decreasing the effectiveness of the proposed protocol. Below, a number of potential methods of dissemination of unsolicited posts in the Aldeon protocol is presented, and the threat they present for network communication is discussed.

Let us assume that the attacker node sends k unsolicited posts. Three general methods of sending them can be distinguished:

- The post sender can create a new conversation branch, and afterwards build more branches extending from it, thus creating a single branch of the conversation containing k unsolicited posts.
- The post sender can create unsolicited posts, which constitute responses to a single post, thus creating k

branches containing unsolicited posts located on one conversation level.

- The post sender can create unsolicited posts, which constitute responses to the following posts in a given branch of the conversation. In this case, assuming that the branch contains k posts, the post sender creates k unsolicited post located on different levels of the conversation tree.

At present, Aldeon allows the removal of unsolicited posts that constitute a single branch. This nullifies the first of the above mentioned attacks by simply cutting a given branch. However, the struggle with the other methods of attacks raises more problems. Both-the second and third of the above mentioned methods require deletion of k posts.

It is to be noted that, to some extent, nodes are able to control the quality of the conversation themselves. If a node deletes a branch of the conversation, it will not be included in the synchronization process. As a result, nodes will not spread posts that they consider to be worthless. However, doing so requires storing the identifiers of the unsolicited posts, which results in slowly accumulating tombstones (deleted messages identifiers).

In order to improve handling unsolicited posts, the following solutions are suggested:

- a lexical filter blocking unsolicited posts based on automatic rules,
- a system of trusted identities, similar to a Web of Trust (this way only the posts signed by trusted authors will make it to the public),
- a closed system of trusted nodes, each employing a well-known spam policy.

5.2. Purposely omitted posts. As shown in Section 3.3, the equality of branch hash function values does not imply the equality of the branches. As a result, two distinct nodes may falsely believe their knowledge on the conversation to be the same. This may also lead to intentional hiding of certain posts, resulting in their omission in the synchronization process.

The post is skipped during the process of synchronization if the value of its branch hash function equals 0. Therefore, it is sufficient to prepare a set of responses to the post in a way that satisfies this condition to enable an attack through deliberately omitting posts. Generating such a set of responses is feasible in the context of Aldeon. However, the threat due to such a form of attack is not as effective as it may seem.

As is known (Dikranjan, 1998), n linearly independent vectors are required to describe an n-dimensional space. This means that, for every vector k in this space, there is a linear combination of vectors that produces k. This property also applies to the collection of vectors summed using $\otimes$ (XOR)—this function is in fact equivalent to adding modulo 2. Additionally, a vector of coefficients for each of the linearly independent vectors is also a member of this n-dimensional space.

In accordance with the above, a set of n posts with linearly independent identifiers allows generating a branch with any desired value of branch hash function. For this purpose, the hash value k of a chosen branch with post that is to be hidden should be written as a linear combination of the input identifiers.

The condition for the feasibility of an attack is therefore the possibility of generation of a set of posts with linearly independent identifiers. It is possible to generate sets of linearly independent identifiers effectively.

Let us assume that there is a space $\mathbb{F}_2^n$. In addition, let there be a non-empty set of linearly independent vectors W, belonging to this space. The set W of these vectors forms a subspace Q of co-dimension k. The probability of selecting randomly an element that does not belong to subspace Q (and is therefore linearly independent of W vectors) is

$$P(x \notin Q) = 1 - 2^{-k}. \tag{27}$$

This means that the mean number of trials required to randomize such an element equals

$$\frac{1}{1-2^{-k}} = \frac{2^k}{2^k - 1}. \tag{28}$$

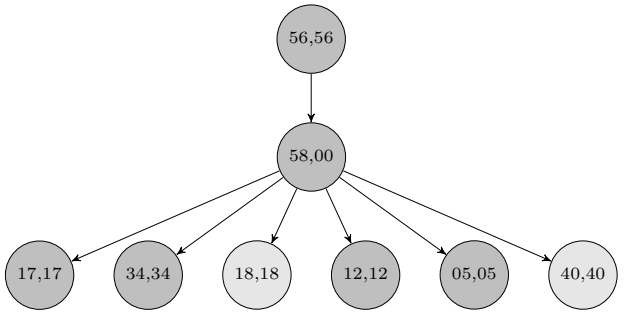
Thus the mean number of trials necessary to obtain a set of n linearly independent vectors equals

$$\begin{aligned} \sum_{k=1}^{n} \frac{2^k}{2^k-1} &= \sum_{k=1}^{n} 1 + \frac{1}{2^k-1} \\ &= n + \frac{1}{3} + \frac{1}{7} + \cdots + \frac{1}{2^n-1} \\ &< n + 2. \end{aligned} \tag{29}$$

It is then possible to use this method to make any post in the topic hidden (make the post have a branch hash function value of 0).

An example of hiding a branch is shown in Fig. 6. In the figure, from a set of 6 linearly independent vectors a subset with any selected sum can be chosen, for example, 58.

Despite the apparent ease of performing an attack associated with hiding the message, it can be shown that its effect is not stable in the context of the Aldeon protocol. The hidden branch will stay undetected as long as the branch structure does not change—for example, by removing any of the added (possibly spam) posts, or by answering any post in the branch. So, the effect of hiding a post will disappear as soon as any node in the network changes the branch structure. Additionally, a



$$S(58) = 58 \otimes 18 \otimes 40 = 0$$

Fig. 6. Example of attack associated with hiding a message.

hidden branch identifier will still be visible as a child in a $Children$ response in the protocol. Therefore, it can be concluded that this attack method has a negligible effect.

6. Performance evaluation

The Aldeon protocol stores messages exchanged between users during their conversations in the form of a tree structure. In this section, we present the results of simulation tests which show how synthetic input data represented by different conversation tree shapes influence protocol optimization. With this end in view, we chose and examined representative tree types that illustrate various possible forms of conversations. Next, we used a real conversation tree structure of a Reddit thread to evaluate the proposed solution in a real-case scenario. The presented simulation experiments were conducted using the PeerSim Java simulation environment (Montresor and Jelasity, 2009).

In the simulations performed and discussed in the paper, two network nodes (called A and B) were used. The scenario of simulations was as follows: first, a tree of a desired type was generated; then, it was copied to both nodes; subsequently, node B added new messages to its tree, thus generating differences between its tree and tree possessed by node A; finally, the Aldeon protocol was launched and node A tried to synchronize with B, which had new posts. The efficiency of synchronization was measured by the number of network messages between A and B (for each request sent by A, B sends one response).

In the performed simulations, four different types of conversation trees were used to represent the initial structures of posts:

- balanced tree (where every post has two children),
- one-level tree (where all posts are on level one and descend directly from the root of the tree),
- list (where every post is related to previous one, from the root to the only leaf),
- furry list (which is a list with additional leafs on every other level).

The chosen conversation trees reflect the structure of real conversations. For example, posts issued in a debate of two users, where each of them has to answer the question, may form a balanced tree. A one-level tree may be constructed when many users comment on selected event. In turn, a list is established when users carry a conversation on a given topic. Finally, a furry list could be an archive of conversations between two users where additional nodes on every level pertain to a specific conversation tree while the main branch represents a list of conversations ordered by their start date.

Once the tree conversation structure was established, it was copied to each node participating in the simulation. Then, by using one of two approaches: leaf-based or uniform distribution, a set number of differences resulting from the addition of new messages was introduced in a conversation tree of one of the nodes. The leaf-based approach allows new messages to be added only to the leafs of a given conversation tree. In turn, in the uniform distribution approach, the probability of adding a new message is equal for each existing node.

All the tests were performed for two sizes of the conversation tree stored in each node (10000 and 100000) to check their influence on the number of network requests. The number of generated differences increases exponentially from one difference to the last value smaller than the tree size (8192 and 65536 for 10 000 and 100 000 nodes, respectively).

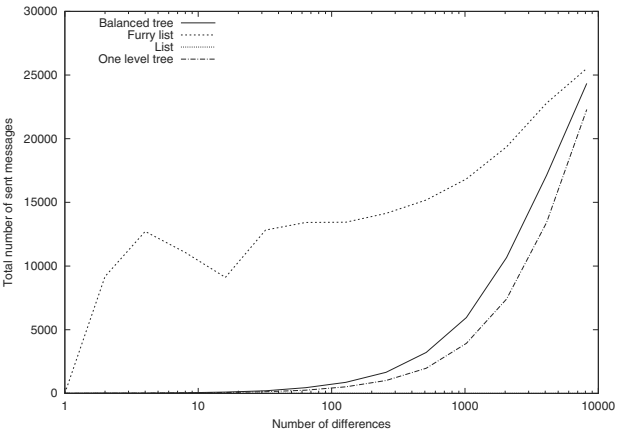


Fig. 7. Number of requests sent during synchronization as a function of the number of differences for the leaf-based difference generation method with 10 000 nodes.

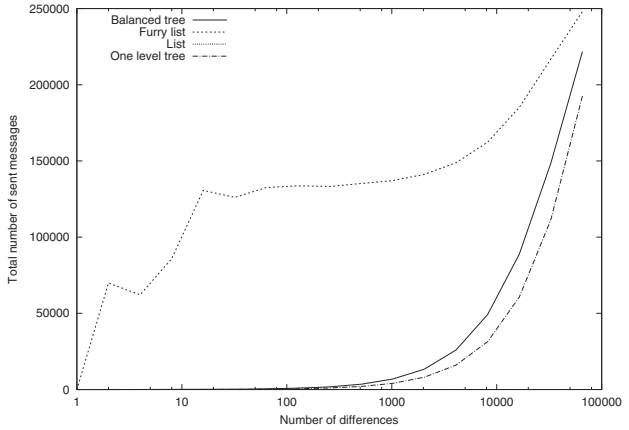


Fig. 8. Number of requests sent during synchronization as a function of the number of differences for the leaf-based difference generation method with 100 000 nodes.

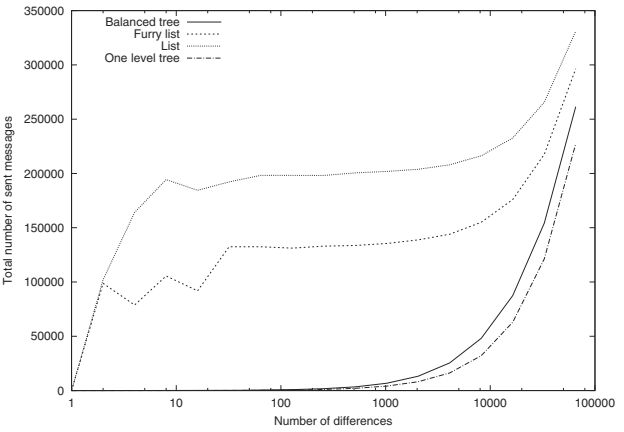


Fig. 10. Number of requests sent during synchronization as a function of the number of differences for the uniform distribution difference generation method with 100 000 nodes.

As can be observed in the presented results (Figs. 7 and 8), Aldeon performs best for a conversation arranged as a list — a stream of replies, one after another, without a single branching path.

In this case, the algorithm requires only four messages for any number of differences. More precisely, one query and response to determine the difference and another pair to confirm that synchronization has ended. Therefore, the function describing performance for the list is $y = 4$. In Figs. 7 and 8 this line is very close to the X axis and thus is barely visible.

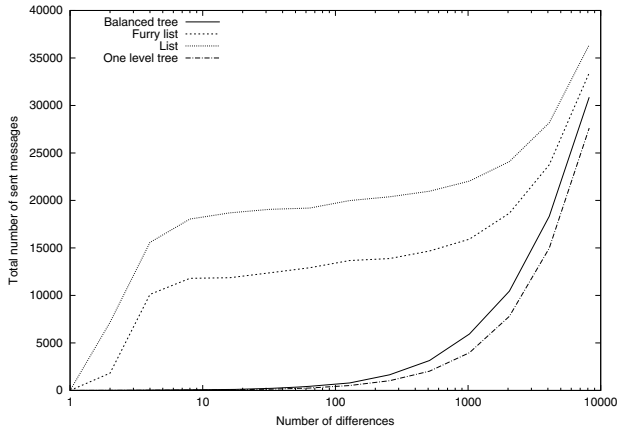


Fig. 9. Number of requests sent during synchronization as a function of the number of differences for the uniform distribution difference generation method with 10 000 nodes.

Of the remaining ones, the furry list fares by far the worst because differences are spread out among multiple levels of the tree and therefore require many synchronization steps to achieve a consistent state. Both the balanced tree and a single node with n children fare on the average, with the number of messages sent increasing proportionally to that of differences found in each tree.

The results obtained using the method of generating differences where each node has the same probability of obtaining new children (Figs. 9 and 10) are similar except for the case of a single list. Here a list conversation structure requires the greatest number of messages in order to synchronize the nodes states, as the messages are no longer contained to a single new subtree, and it is necessary to traverse the entire list.

According to the results presented above, it is hard to indicate the messages structure whose application would result in always optimizing the synchronization process. For example, a list structure, which is adopted as the default message structure by many peer-to-peer group communication and instant messaging solutions available on the market (described in Section 7), gives good results when the new posts are added at the end of the list, and the number of conversation participants is relatively small. Meanwhile, in real applications, conversations may take many forms (for example, Reddit conversations take the tree form), and posts may be added by conversation participants in random locations in the conversation structure (which implies that with a large number of users the conversation structure could be chaotic).

Consequently, in subsequent simulations we used the parsed tree structure of a real Reddit thread (2251 posts) as input data for the Aldeon protocol. The obtained information consisted of the posts tree structure and post creation timestamps.

In order to measure the performance gains resulting from the usage of the branch hash function, in the following simulations we compared two approaches. The first was the proposed BHF-based algorithm, and the second was the same algorithm with disabled *Suggest* messages. Thus, the second algorithm is identical to the naive DFS-based approach (i.e., the approach in which all messages are sent one by one while visiting tree nodes, with deterministic hashes for all branches). We compared the number of request/response pairs sent during the synchronization process between two nodes. It must be stressed that most requests were sent in parallel—there will be at most as many causally related requests as the height of the tree.

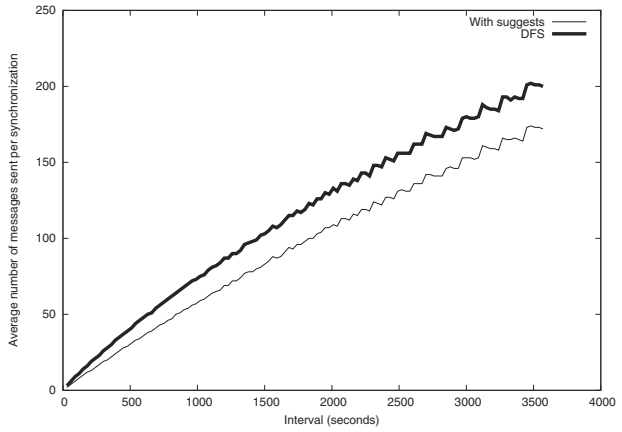


Fig. 11. Performance improvement obtained using the $Suggest$ messages.

The post trees stored on the nodes before the synchronization consist of all posts published prior to a chosen timestamp. The difference at chosen timestamps (interval) affects the similarity of the tree structures—the shorter the interval, the fewer the differences in trees. Performance improvement was expected for uneven distribution of differences as the BHF facilitates finding those changes. We measured the number of requests sent as a function of the interval. We divided the entire thread timespan (ca. 18 hrs) into time intervals, ranging from 30 to 3600 seconds. In order to compute performance for a given interval length we average the number of requests sent in synchronization procedures in all intervals of that length.

We can see in Fig. 11 that the algorithm incurs no penalty in terms of network traffic—as long as the identifiers are long enough to prevent hash collisions, the $Suggest$ messages always correctly point to a tree difference, skipping multiple tree levels.

As expected, the algorithm benefits from uneven distribution of differences in the trees (as is the case when the trees are similar), and gradually approaches the normal method performance as the differences become more and more evenly distributed, which is shown in Fig. 12. Using the BHF, we achieved up to a 30% reduction in the number of sent messages.

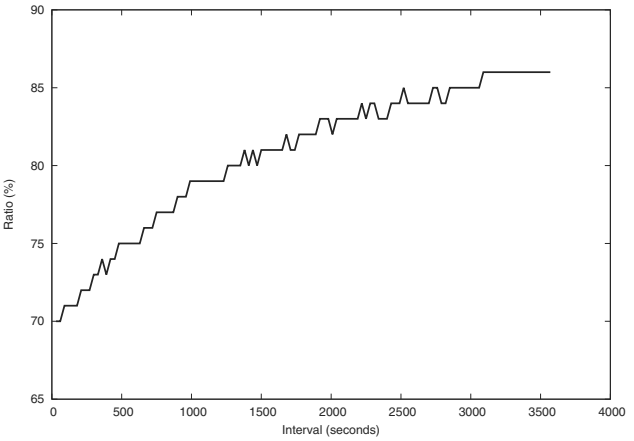


Fig. 12. Ratio of an average number of requests sent with the $Suggest$ messages and with the DFS-based approach.

7. Related work

Recently, a few ideas for creating group communication and instant messaging tools using peer-to-peer systems have appeared (Mannan and van Oorschot, 2006; Serjantov, 2002; Berthold *et al.*, 2001; Rowstron *et al.*, 2001). The proposed solutions aim to help small, medium and corporate offices by enhancing business productivity and internal communication. Common functionalities include taking notes, setting reminders, remote desktop sharing, file transfer, instant messaging, broadcasts, private and group communication. Below, we present the advantages and disadvantages of a representative subset of solutions, which emerged from both academic and business backgrounds. The analysis of chosen solutions takes into account a number of criteria. It is discussed whether message integrity is ensured, and whether its content is encrypted. Another criterion is protecting the identity of message senders and receivers. Furthermore, it is checked if the solutions are able to handle conversations between multiple users. In addition, the time in which the recipient is required to receive the message is taken into account. Moreover, data structures used by the solutions considered to store conversations are presented. The data structures include a list, a tree, and a partially ordered set. In Table 1, the latter is referred to as a P-O set. Finally, we discuss whether a solution supports migrating to other devices or not.

Bitmessage (Warren, 2012) is meant to be an

alternative to insecure email. While well-known secure alternatives exist (PGP/GPG), they are not easy to use. Bitmessage is a decentralized, trustless peer-to-peer communication protocol. Each user has a pair of cryptographic keys, and the public key serves as the user identifier. Each message is signed by the senders' private key (making the message content non-repudiable) and encrypted by the receivers' public key (thus only the receiver having a matching private key is able to decrypt it). The message dissemination mechanism in this P2P system is based on Bitcoin's transaction and block transfer system (Nakamoto, 2008; Sompolinsky and Zohar, 2013; Mooser *et al.*, 2014). There are several important consequences of following the Bitcoin way.

To send one message, a proof of work (a concept from Bitcoin) with average computation time of four minutes has to be completed. A message with a proof of work is propagated by other nodes on a best effort basis, in order to be received by each node in the network. The authors of Bitmessage suggest that, since every message is sent to each node anyway, it would be natural to have a broadcast mechanism, although they do not say how exactly it would work (how the message would be encrypted or how it would work with the concept of streams—explained later). From the above it can be concluded that Bitmessage provides the following benefits: decentralization, encryption of messages, masking sender and receiver identities, an unspoofable sender. An additional advantage of Bitmessage is an active community around it.

Several extensions/tools enhancing Bitmessage have been proposed (for example, Mailchuck—a bridge enabling communication with other people using email). Despite the relatively high popularity, Bitmessage has also a few disadvantages. Each node which received a message has to try to decrypt it with its private key, because there is no faster way to determine whether this node was the intended recipient. Furthermore, it is possible for a node not to receive a message if it is absent from the network for more then 48 hours. The scalability of this solution is questionable, since each node has to store all messages in the system. The Bitmessage authors propose to alleviate this problem by keeping each message for only 48 hours and by introducing streams. The basic idea is to divide the network into separate subparts and disseminate messages only in these smaller subparts. Streams form a hierarchy—each stream may have up to two child streams. Each node in the parent stream maintains a list of few peers in the child streams. Another solution to the problem of securely sending and receiving messages in a peer-to-peer environment is ShadowCoin Secure Messaging (also known as ShadowChat) (SDCDev, 2014). It is a peer-to-peer encrypted instant messaging system, designed to protect user privacy, which was developed as a supplementary tool for ShadowCoin, an anonymous cryptocurrency.

In principle, ShadowChat is similar to Bitmessage (it is close to being called a clone). Still, there are some differences; for example, messages are not encrypted with the recipient's public key but with a symmetric AES-256-CBC algorithm (Frankel *et al.*, 2003). The secret key in ShadowChat is shared with the use of the elliptic curve Diffie–Hellman method (ENISA, 2012), and the messages are signed with the elliptic curve digital signature algorithm (ECDSA). Messages in ShadowChat are distributed over the existing ShadowCoin peer-to-peer network. A copy of each encrypted message is stored on a ShadowChat node for 48 hours, and then deleted (as in Bitmessage), but there is no mention of streams. Messages are stored in the system in the form of a partially ordered set. The stored messages are grouped, so the system operates on groups of messages, thus saving the bandwidth. Among the benefits offered by ShadowChat are faster message dissemination and less CPU intensive operation (no proof of work needed for messages), optimizations for bandwidth use. Unfortunately, it shares most of Bitmessage drawbacks.

Another solution worth mentioning is Bleep—a secure P2P-based instant messaging application. Bleep uses a public key instead of a login name, which makes it possible to protect the privacy of communication by hiding the identity of its participants. Each Bleep user has a keypair that he/she may register on a central server, associating it with his/her email or phone number, making it easier for other users to find him/her. Messages are encrypted using the private key of the sender and the public key of the receiver. Bleep supports forward secrecy, which means that even if a private key were to be compromised it would not affect any future traffic. For each conversation, a new keypair is generated, which is then transformed using a one-way function for each message in the conversation, so every subsequent message is encrypted using different key, with the previous one being deleted. The main advantage it has over other P2P-based chats is that it does not need a central server for message routing. Lookup on the server is done only once per added contact to find the public key associated with the account. After that, users only apply the DHT to find each other's IP address and establish an encrypted tunnel over UDP between them. An additional interesting feature of Bleep is that it allows its users to make voice calls in addition to instant messaging.

Finally, CryptoCat (Berthold *et al.*, 2001) has to be mentioned. This solution consists of two primary elements: a web-based chat application (typically loaded via a browser extension) and the CryptoCat protocol for an encrypted group chat. It is a client-server solution, added here for comparison. The future plan for CryptoCat is to switch from a custom protocol to using mpOTR (multi-party off-the-record) (Payne and Edwards, 2008).

Table 1. Summary of related solutions.

	Bitmessage	ShadowChat	Aldeon	Bleep	CryptoCat
Protects sender's identity	Y	Y	Y	Y	Y
Ensures message integrity	Y	Y	Y	Y	Y
Protects receiver's identity	Y	Y	N	Y	Y
Encrypts message content	Y	Y	N	Y	Y
Many recipients	Y	N	Y	N	Y
Msg has to be received in	48 hours	48 hours	unbound	instantly	instantly
User can switch devices	N	N	Y	Y	N
Structure of messages	list	P-O set	tree	P-O set	P-O set
Logical conversation order	N	N	Y	N	N

CryptoCat currently uses OTR for a chat involving only two users. The advantage of this application is a very low barrier to entry—it is enough just to open the web page or install the browser extension. The two frequently mentioned CryptoCat disadvantages are no support for offline messages and limited security, resulting from public key authentication and unstable user identities.

Table 1 compares selected features of the solutions presented above. All solutions protect the sender identity by using public/private keys. The sender cannot be spoofed, because messages are signed with private key (which is difficult to forge). Obviously, it a has to be protected from being stolen. Furthermore, a public key of the sender is not registered at any central repository, nor is it connected to other contact information such as email or telephone number. Bleep is an exception—a user may provide this information (but it is not required). Message integrity is protected as well in all solutions, by cryptographic signatures. Every solution, except for Aldeon, protects the receiver identity and encrypts messages. It is the first major difference between Aldeon and the other solutions, which aim to help people have discussions in a closed, private environment. Aldeon, on the other hand, focuses on open, anonymous conversations similar to Internet forums or imageboards, which may have many participants. That is why there is no need to protect the identity of the receiver or to encrypt messages. Aldeon naturally enables multiple recipients, since every interested party is free to read a conversation. There is a mention of broadcast functionality in the Bitmessage whitepaper, but it is not properly explained. CryptoCat supports group conversations. There is no mention of this type of feature in data about ShadowChat or Bleep. Bitmessage and its clone—ShadowChat introduce the limitation of 48 hours to receive a message. Bleep and CryptoCat require that both parties be online during conversation. Aldeon requires only that at least one person interested in the conversation (not necessarily writing messages in it) be online. Another feature, distinctive for the proposed solution, is the structure in which messages exchanged between conversation participants are stored. Only Aldeon uses a tree to represent conversations. Each message in a conversation is sent in response to another one. That relationship is represented by a directed edge in the tree. Therefore, using this structure gives the benefit of preserving the logical order of messages. Out of all solutions, only Bleep supports migrating to other devices. In Aldeon, on a new device it is enough to copy the private key to be able to issue new messages. The old messages can be easily retrieved if the user provides identifiers of the topics.

8. Conclusions

This paper introduced the Aldeon protocol for group conversations in peer-to-peer systems. Fast propagation of messages among all interested peers is achieved in the proposed solution by combining the DHT and a novel branch hash function synchronization algorithm, which significantly reduces synchronization time even for very large trees. Moreover, the system model we adopted in Aldeon and the decentralized identity management scheme significantly improve the overall safety of users. The obtained experimental results showed that the proposed solution is efficient in the case of real discussions between multiple conversation participants.

Our future work encompasses the introduction of a user defined list of trusted nodes. Consequently, during the bootstrap phase, the information would be obtained only from such nodes. Additionally, it would be possible to opt into exchanging posts exclusively with nodes on that list.

An important direction of future protocol extension is designing an anti-spam mechanism. Introduction of a lexical filter would enable nodes to detect unwanted messages before they are stored locally.

Another plan for development is extending, currently one-sided, the synchronization mechanism with bilateral synchronization. If it were implemented, node A downloading posts from node B would be able to simultaneously offer posts in its (A's) possession which B lacks.

References

Aditya, P., Erdélyi, V., Lentz, M., Shi, E., Bhattacharjee, B. and Druschel, P. (2014). Encore: Private, context-based communication for mobile social apps, *Proceedings of the 12th Annual International Conference on Mobile Systems, Applications, and Services, Bretton Woods, NH, USA*, pp. 135–148.

Baruah, T. (2012). Effectiveness of social media as a tool of communication and its potential for technology enabled connections: A micro-level study, *International Journal of Scientific and Research Publications* **2**(5): 1–10.

Berthold, O., Federrath, H. and Kpsell, S. (2001). Web mixes: A system for anonymous and unobservable internet access, *International Workshop on Designing Privacy Enhancing Technologies: Design Issues in Anonymity and Unobservability, Berkeley, CA, USA*, pp. 115–129.

Boyd, C., Mao, W. and Paterson, K.G. (2005). Deniable authenticated key establishment for internet protocols, *in* B. Christianson *et al.* (Eds.), *Security Protocols*, Springer, Berlin/Heidelberg, pp. 255–271.

Dabek, F., Li, J., Sit, E., Robertson, J., Kaashoek, M. and Morris, R. (2004). Designing a DHT for low latency and high throughput, *Proceedings of the 1st USENIX Symposium on Networked Systems Design and Implementation (NSDI'04), San Francisco, CA, USA*, p. 7.

Damiani, E., De Capitani di Vimercati, S., Paraboschi, S. and Samarati, P. (2004). P2P-based collaborative spam detection and filtering, *4th International Conference on Peer-to-Peer Computing (P2P 2004), Zurich, Switzerland*, pp. 176–183.

Dikranjan, D. (1998). Recent advances in minimal topological groups, *Topology and its Applications* **85**(1): 53–91.

ENISA (2012). Algorithms, key sizes and parameters report—2013 recommendations, *Technical report*, European Union Agency for Network and Information Security Agency, Heraklion.

Frankel, S., Glenn, R. and Kelly, S. (2003). The AES-CBC cipher algorithm and its use with IPSEC, *RFC 3602*, Network Working Group, `https://tools.ietf.org/html/rfc3602`.

Gilbert, S. and Lynch, N. (2002). Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services, *SIGACT News* **33**(2): 51–59.

Laprie, J.C., Avizienis, A. and Kopetz, H. (Eds.) (1992). *Dependability: Basic Concepts and Terminology*, Springer-Verlag New York, Secaucus, NJ.

Lv, X., Li, H. and Wang, B. (2012). Group key agreement for secure group communication in dynamic peer systems, *Journal of Parallel and Distributed Computing* **72**(10): 1195–1200.

Mannan, M. and van Oorschot, P. (2006). A protocol for secure public instant messaging, *10th International Conference on Financial Cryptography and Data Security, Anguilla, British West Indies*, pp. 20–35.

Miller, K. (2014). *Organizational Communication: Approaches and Processes*, Cengage Learning, Boston, MA.

Montresor, A. and Jelasity, M. (2009). PeerSim: A scalable P2P simulator, *IEEE 9th International Conference on Peer-to-Peer Computing, P2P'09, Seattle, WA, USA*, pp. 99–100.

Moore, A. and Zuev, D. (2005). Internet traffic classification using Bayesian analysis techniques, *Proceedings of the International Conference on Measurements and Modeling of Computer Systems, SIGMETRICS 2005, Banff, Alberta, Canada*, pp. 50–60.

Mooser, M., Boohme, R. and Breuker, D. (2014). Towards risk scoring of Bitcoin transactions, *Financial Cryptography and Data Security FC 2014 Workshops, Bridgetown, Barbados*, pp. 16–32.

Nakamoto, S. (2008). Bitcoin: A peer-to-peer electronic cash system, `www.bitcoin.org`.

Payne, B. and Edwards, W. (2008). A brief introduction to usable security, *IEEE Internet Computing* **12**(3): 13–21.

Rivest, R. L., Shamir, A. and Adleman, L. (1978). A method for obtaining digital signatures and public-key cryptosystems, *Communications of the ACM* **21**(2): 120–126.

Rowstron, A., Kermarrec, A., Castro, M. and Druschel, P. (2001). SCRIBE: The design of a large-scale event notification infrastructure, *Proceedings of the 3rd International COST264 Workshop on Networked Group Communication, NGC'01, London, UK*, pp. 30–43.

Sakarindr, P. and Ansari, N. (2010). Survey of security services on group communications, *IET Information Security* **4**(4): 258–272.

Saxena, A., Misra, J. and Dhar, A. (2014). Increasing anonymity in Bitcoin, *in* N. Christin and R. Safav-Naini (Eds.), *Financial Cryptography and Data Security*, Springer, Berlin/Heidelberg, pp. 122–139.

Schoder, D. and Fischbach, K. (2003). Peer-to-peer prospects, *Communications of the ACM* **46**(2): 27–29.

Schollmeier, R. (2001). A definition of peer-to-peer networking for the classification of peer-to-peer architectures and applications, *Proceedings of the 1st International Conference on Peer-to-Peer Computing, P2P'01, Linköping, Sweden*, pp. 101–102.

SDCDev (2014). Shadowcoin secure messaging: A P2P encrypted instant messaging system, `www.shadowcoin.co`.

Serjantov, A. (2002). Anonymizing censorship resistant systems, *in* P. Druschel *et al.* (Eds.), *Peer-to-Peer Systems*, Lecture Notes in Computer Science, Vol. 2429, Springer, Berlin/Heidelberg, pp. 111–120.

Sit, E. and Morris, R. (2002). Security considerations for peer-to-peer distributed hash tables, *Revised Papers from the 1st International Workshop on Peer-to-Peer Systems, IPTPS'01, Cambridge, MA, USA*, pp. 261–269.

Sompolinsky, Y. and Zohar, A. (2013). Accelerating Bitcoin's transaction processing. Fast money grows on trees, not chains, *IACR Cryptology ePrint Archive* **2013**: 881.

Völker, L., Noe, M., Waldhorst, O.P., Werle, C. and Sorge, C. (2011). Can internet users protect themselves? Challenges and techniques of automated protection

of http communication, *Computer Communications* **34**(3): 457–467.

Warren, J. (2012). Bitmessage: A peer-to-peer message authentication and delivery system, `www.bitmessage.org`.

Weninger, T. (2014). An exploration of submissions and discussions in social news: Mining collective intelligence of Reddit, *Social Network Analysis and Mining* **4**(1): 1–19.

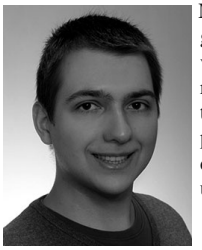
Xiao, Z., Guo, L. and Tracey, J. M. (2007). Understanding instant messaging traffic characteristics, *27th IEEE International Conference on Distributed Computing Systems (ICDCS 2007), Toronto, Ontario, Canada*, p. 51.

Xie, M. and Wang, H. (2012). Secure instant messaging in enterprise-like networks, *Computer Networks* **56**(1): 448–461.

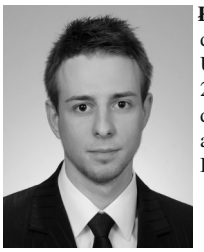
Zhang, Q., Cheng, L. and Boutaba, R. (2010). Cloud computing: State-of-the-art and research challenges, *Journal of Internet Services and Applications* **1**(1): 7–18.

Anna Kobusińska received her M.Sc. and Ph.D. degrees in computer science from the Poznań University of Technology in 1999 and 2006, respectively. She currently works as an assistant professor at the Laboratory of Computing Systems, Institute of Computing Science, Poznań University of Technology. Her research interests include large scale distributed systems and algorithms, big data, SOA, fault-tolerance, replication and consistency models.

Jerzy Brzeziński received an M.Sc. in electrical engineering, and a Ph.D. and D.Sc. in computer science, all from the Poznań University of Technology, where he is currently a full professor of computer science. His research interests include distributed algorithms and fault-tolerant distributed systems. He is the author and a coauthor of two books, and over 100 research papers published in journal and proceeding of many international conferences. He has been involved in many international and national research projects. Prof. Brzeziński is a member of the IEEE CS, ACM, Polish Information Processing Society, Computer Science Committee of the Polish Academy of Sciences, and others.



Michał Boroń received the B.Sc. and M.Sc. degrees in computer science from the Poznań University of Technology, Poland, in 2014 and 2015, respectively. His research interests include distributed algorithms, P2P systems, and cloud computing. He is currently employed as a software developer at Roq.ad (EU leader in cross-device user identification for digital advertising).



Łukasz Inatlewski received his B.Sc. and M.Sc. degrees in computer science from the Poznań University of Technology, Poland, in 2014 and 2015, respectively. His research interests include database management systems, machine learning and data mining. He is currently employed as a IT professional at Roche.

Michał Jabczyński is a graduate of the Poznań University of Technology, Poland. He received a B.Sc. degree in 2014 and an M.Sc. in 2015, while he was also studying mathematics at Adam Mickiewicz University in Poznań. He is an intern at Google Paris, and will start a Ph.D. at UPMC in late 2016. His research interests include weak consistency database models and blockchain-based consensus.

Mateusz Maciejewski received his B.Sc. and M.Sc. degrees in computer science from the Poznań University of Technology in 2014 and 2015, respectively. His research interests include distributed algorithms, P2P systems and procedural content generation.

Received: 12 January 2015
Revised: 26 July 2015
Re-revised: 7 December 2015
Accepted: 12 January 2016