A Generic Data Exchange System for Friend-to-Friend Networks

Cyril Soler

RESEARCH REPORT
N° 9107
Oct 2017
Project-Team Maverick

ISSN 0249-6399   ISRN INRIA/RR--9107--FR+ENG

A Generic Data Exchange System for Friend-to-Friend Networks

Cyril Soler

Project-Team Maverick

Research Report n° 9107 — Oct 2017 — 22 pages

Abstract: Decentralized private networks (a.k.a. darknets) guaranty privacy and concealment of information against global observers. Although a significant number of decentralized data distribution systems ex- ist, most of them target peer-to-peer architectures where any pair of nodes can exchange data using a temporary encrypted connec- tion. Little has been done to achieve the same confidentiality in static darknet architectures, also known as "Friend-to-Friend" net- works, in which participants form a static mesh of nodes, each node only talking to a set of "friend" nodes managed by the user himself. Distributing data over F2F networks requires a specific model of confidentiality and proper algorithms in order to seamlessly spread the information beyond immediate friends nodes. In this paper we present a secure, robust and generic data distribution system that is specifically suitable to F2F networks. The proposed system manages pseudo-anonymous identities grouped into circles, that can be used to limit the access to information. It offers built-in reputa- tion control and an abstract data layer to represent the information to spread. On top of that we present several existing applications to that system, providing the functionality of distributed forums, asynchroneous messaging, distribution channels, and also discuss the possibility to create social networking applications on top of it.

Key-words: Decentralized Online Social Networks, Friend-to-Friend Networks, Darknets

Un Système de Distribution Robuste et Sécurisée sur les Réseaux F2F

Résumé : Les réseaux privés décentralisés (ou Darknets) garrantissent la securite des informations contre l'espionnage de masse. Bien qu'une quantité significative de réseaux privés décentralisés existe dans le monde de l'*open source*, ceux ci sont généralement basés sur une architecture "pair à pair" (P2P) dans laquelle des communications directes sont succeptibles d'intervenir entre toutes les paires de noeud du réseau. Nous nous interessons plus particulièrement à la distribution sécurisée de données dans les réseaux de type "ami à ami" (F2F), pour lesquels chaque noeud du réseau ne communique directement qu'avec une liste pré-déterminée de noeuds amis controllée par l'utilisateur. La distribution sécurisée de données sur ce type de réseau demande des algorithmes et un modèle de confidentialité adaptés adaptés Dans ce rapport, nous proposons un système robuste, sécurisé et extensible de distribution de données sur les réseaux F2F, proposant des identités pseudo-anonymes controllées par un système de réputations, et pouvant être regroupées en cercles d'amis pouvant être utilisés pour limiter la distribution des informations. Nous présentons plusieurs applications basées sur ce système, implémentant des forums, des chaines de distribution ainsi qu'une messagerie asynchrone.

Mots-clés : Réseaux Sociaux Décentralisés, Réseaux F2F, Darknets

1 Introduction

Decentralized online social networks have recently come been under focus, especially after the scandals of global surveillance highlighted by the Snowden leaks. Many decentralized OSNs are available today, most of which are based on a P2P architecture [PFS14] where nodes can freely exchange data with any other nodes. In this paper, we focus on secure and private data distribution over a specific subclass of online social networks, called "Friend-to-friend" networks (or darknets), in which each network node only exchanges data with a designated list of "friend" nodes—generally managed by the user who owns the node.

Friend-to-friend networks (further noted F2F) come with interesting intrinsic properties: **2-hops anonymization:** information beyond friend nodes can easily be guarantied to be anonymous. Since data always come from your friend nodes, only these nodes can see your own IP address (if that address is made visible by the friend-to-friend link); **local web of trust:** friend nodes can team up to provide data integrity and control; **local distributed caching:** friend nodes can be used to cache some shared information while the node is offline; **consistent local interest:** friend nodes that are selected on the basis of existing real-life relationships can be trusted to provide relevant/interesting content.

However, F2F network architectures also raise specific challenges for a secure distribution of data: **no global addressing:** in F2F networks, each node only has a local knowledge of network topology, and cannot talk directly to a non friend node. Distributed information therefore needs to consistently spread from friend to friend. This poses serious difficulties when it comes to ensuring data availability [GTAL12]; **global consistency:** data distribution needs to be fast enough and flexible. In a system that provides grouping of users for instance, adding and removing group members should rapidly take effect over the whole reachable network; **network heterogeneity:** the bandwidth load should be as light as possible because the number of friend connections is usually quite limited (the order of magnitude is tens), although it can locally be very large too (several hundreds); **robustness:** data distribution needs to be robust to random disconnection of friend nodes. Some amount of redundancy is therefore needed, but shouldn't result into wasting network bandwidth; **security against non friend nodes:** data distribution needs to be safe against rogue network nodes that may attempt to propagate of unwanted/corrupted data, or implicate target users in unwanted activities, or simply attempt to flood the system. Data should also be protected against errors in transmission; **privacy against non friend nodes:** the distribution of data should not disclose any information about the topology of the network beyond friend nodes. Users should be given the possibility to remain anonymous while providing authentication and control over their own data.

Our contribution is a Generic data eXchange System (hereby named GXS) for friend-to-friend networks that leverages the properties of F2F networks while ensuring a high degree of privacy and security. In this system, friend nodes help each other to maintain data consistency and ensure distribution, while relying on cryptography for privacy and authentication. The GXS system only requires the underlying mesh of network nodes to provide pairwise authenticated/encrypted links between friend nodes, and a cryptographic signing primitive to optionally connect some of its own data to the owner node. On top of the F2F mesh, GXS ensures the distribution of data, while privileging by design the information that most interests users around each node and ensuring a high degree of redundancy so that all communications can be kept asynchronous. GXS provides transport for a generic definition of data that can be adapted to represent very different types of content, thereby offering a high degree of extensibility.

The notion of security is central to the GXS system so as to ensure both integrity, authentication/anonymity (depending on the requirements of the service it is used in) and privacy of the data. While the hosting darknet architecture protects against global observers, GXS authentication and encryption system prevents participants in the network to access data that is not meant for them. In addition each network node locally encrypts its data making it theoretically unreachable to physical access when a node is not running. The GXS system guaranties both anonymity and authentication of data using pseudo-anonymous identities that are optionally linked to the hosting network node. These identities can be grouped into circles, which in turn can be set to limit the distribution of other data. Generally speaking, GXS ensures security by design rather than software control. This means that users are denied access to private data not because friend nodes refuse to send it, but because encryption makes it unreadable.

We present in this paper several services based on GXS that already run on top of an existing F2F network: distributed forums where users can exchange ideas in a threaded hierarchy of topics; information channels where a restricted set of users can post authenticated data accessible to wider set of users who can react/respond using threaded comments; an asynchronous messaging service that allows pseudo-anonymous identities to robustly exchange emails. These services are currently used by thousands of users everyday in an existing F2F network used as a test bed, but we believe that GXS is modular enough to be fit into any existing darknet architecture. We also discuss in this paper the possible implementation of fully decentralized versions of existing social network services similar to Facebook, Twitter and GitHub on top of F2F networks using GXS.

2 Related Work

Cryptographic primitives for decentalized data distribution. Attribute-Based Encryption allows to encrypt data for groups following a given *access structure* that allows an accurate definition of who can access which data [BSW07]. ABE operations are 100-1000 times slower than those of RSA, which we think is absolutely prohibitive. ABE secures information to groups of users by defining and distributing a symmetric key for each group. Every information is encrypted using this symmetric key. This key needs to be updated at every changes in a group, which makes the synchronizing of group membership challenging over a large network. Some social network systems however have been designed based on ABE systems: Persona [BBS$^+$09, JNM$^+$12] uses ciphertext policy attribute based encryption and creates lists of users with access rights. Further instances of this system such as DECENT [JNM$^+$12] and–an update–Cachet [NJM$^+$12], rely on a DHT to store the encrypted data. They implement an improved version of ABE that allows immediate revocation of attributes, and provides a fine grained management of access control.

Predicate Encryption [KSW08] allows to tune the access policy for each piece of data using attribute based policies. Shared secret keys are created for each pair of friends which they use to decrypt the data on each other's profiles. Bodriagov [BKB14] proposes a system to further hide users access policies. Günter *et al.* [GMS12] base the distribution on broadcasting encrypted data with pseudonyms. In his system, an attacker can still figure out which pseudonym that can actually decrypt the data. Anderson et al. [ADBS09] present an information distribution system suitable to social networks in which a hiearchical structure of symetric keys provides selective access to users' profile data.

Taohe [WOW08] is a distributed file sharing system. Encrypted files are made accessible by sharing a pair of keys that are used for authentication and encryption of the file. This system requires a lot of overhead in order to share the keys for each file, which makes it hardly suitable for a large social network with a fine grain of information sharing.

The GNU name system [Wac15] provides a secure replacement for DNS over the GNUnet network, which allows to publish links to various types of data inside the network. Information is linked to identities represented by asymmetric keys. GXS also uses asymmetric keypairs, but these are not directly related to the node that hosts them, offering the possibility to maintain full anonymity of the actors.

Operations in GXS are generally simpler (and faster) than attribute based encryption methods and predicate encryption, because GXS leverages the rather static topology and trusted connections in the F2F network.

Decentralized secure distribution systems. There is a large list of decentralized Online Social Networks (OSN). See [PFS14] and[ZI16] for a detailed survey. Diaspora [dia] builds a decentralized social network with two levels of storage: master nodes acting as data servers and user nodes acting as clients.

PeerSon [BSVD09] uses a key sharing/distribution system to control the access of data into a decentralized social network. The proposed architecture relies on a DHT. It does not hide metadata nor IPs. It does not provide dynamic group membership either. SafeBook [CMS09] uses a little less decentralized architecture in which concentric structures in the network layer provide storage and privacy for the participating network nodes. LotusNet [AR12] has a fully P2P network model and therefore relies on an authenticated DHT to establish connections and retrieve data, while also offering reputation management. Pythia [NAHK11] is a decentralized architecture that aims at providing social search. Finally, a social messaging service has been implemented inside GNUnet [Tot13] in the goal of providing a social network layer to the GNUnet system.

Most existing systems rely on a dynamic P2P network (as opposed to a F2F darknet) and therefore need a DHT to either store or at least index the data, and establish connections to who owns it. Conversely, the friend-to-friend allies a rather static network topology and probable common interest of friends, which our system leverages for distributing the data and ensuring availability, while locally distributing the information among friend nodes.

Data availability in friend-to-friend networks. An empirical Study of availability in F2F Systems [SDDM11] shows that the problem of optimizing availability with minimum content replication is NP-complete. The difficulty of reliably providing information has been shown to highly depend on correlation of data availability [GTAL12]. Pietinanen [PD12] studies the impact of temporal communities on the efficiency of opportunistic content dissemination in friend-to-friend networks. In this paper, our assumption that friends share common interests for for the same types of data somewhat forces this correlation by design.

In [TMB+13] the authors covers the efficiency of cooperative caching in distributed social networks, trying to minimize network-wide content provisioning cost. The *Psephos* system [IMC10] provides distributed caching for mobile networks. The system is designed to be efficient despite heacy network heterogeneity. Caching strategy is based on a voting system that allows to maintain local information about how much an object is important in the network. In [TVSP17] the authors present a community based greedy algorithm that targets mobile networks primarily, exploiting the possibility of content replication while trying to minimize content replication and data traffic, at the expense of data availability and speed of delivery. In contrast to these systems that try to reduce storage based on a global optimization strategy, our system bases storage on users' interest.

In practice there only exists a few established friend-to-friend networks. We believe that the need to find entry points to a existing network and the need for users to manage their connections explicitly restrains the adoption of these solutions. OneSwarm [IPKA10] was mainly dedicated to file transfer and the project is now dead, GNUnet [Wac15] and Freenet [CSWH01] can both be used as F2F networks if the appropriate options are set by the user but still run as open networks so their F2F usage is likely to not be broadly adopted. Retroshare [Sol17, JFH] is also a friend-to-friend network which offers both file sharing and decentralized services such as forums. Although there is no reliable statistics about the number of users, this almost 10-years old software has been downloaded more than 500,000 times. Our distribution system has been implemented and runs over the Retroshare network(s).

3 Overview

The GXS distribution system is designed to work on top of friend-to-friend darknets, which means that computers running GXS instances (referred to as *nodes* in this paper) are linked together by encrypted connexions. These connexions are established using TLS, signed by PGP certificates. The network topology is driven by the list of node certificates each user allows his node to connect to.

The network is allowed to be very inhomogeneous both in shape and bandwidth: the number of friend nodes to a given node typically varies from 1 to hundreds, and bandwith is rather asymmetric since a lot of users typically rely on asymetric DSL connexions.

Although nodes can connect/disconnect in an unpredictable way, we assumed that the data stored at friend nodes is rather

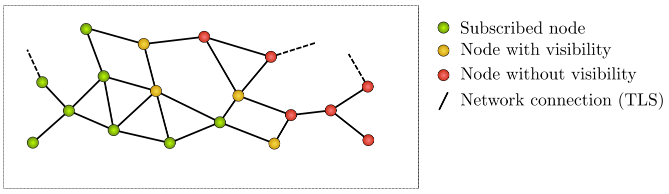


Figure 1: Distribution paradigm for GXS data: green nodes are subscribed to a given set of GXS data (think of e.g. a forum). Subscribed nodes advertise the existence of that data to their own friends (Yellow nodes), who can then decide to subscribe and therefore make the data more widely accessible in the network. Nodes in red cannot see that data.

consistent over time, so that the distribution of data can be based on synchronising the difference between what's stored at neighbour nodes.

The data distribution paradigm of GXS is that nodes "subscribe" to interesting information. Doing so they advertise the existence of this information to their own neighbor nodes. This is illustrated in Figure 1. This model inherently promotes interesting data throughout the network. The design of GXS addresses the following challenges:

- Integrity, authentication/anonymity and privacy of the distributed data, depending on the specific needs of the target application;
- synchronization with minimal bandwidth overhead while promoting interesting data;
- robustness to changes in the network topology
- robustness to spam;
- extensibility.

In this paper we present the architecture of the GXS system as well as a few use cases applications (distributed forums, publication channels, e-mail,...) to illustrate its generic nature.

4 The Generic eXchange System

We successively describe how data is represented, authenticated, stored, distributed, and how applications combine the various elements to create functionality.

4.1 GXS Objects

The GXS system relies on 5 main data components: identities, circles, services, groups and messages.

Identities are pairs of asymetric keys (RSA in our implementation) used to author the distributed data, at the level of a single user. Identities can be anonymous, in which case they only provide the guarranty that two pieces of data have the same author; they can also be signed by a network node key, in which case they explicitely connect the data to a specific node of the underlying F2F network.

Circles are set of *identities*, that can be used to restrict the distribution of data.

Services represent functionality. A GXS service is a component that works as interface between a specific application (e.g. distributed forums) and GXS internal objects such as *groups* or *messages*. Each service has its own data format and storage database.

Groups are the top level type of data that a given service may use. A service may create as many groups as needed, each having its own authentication and publish policies. In the example of distributed forums, each forum is a GXS *group* which may be restricted to some specific *circle* of *identities*.

Group Meta Data	
Field	**Type**
Group Id	128 bits fingerprint of the public admin key
Publish time	32-bits integer
Circle Id	Group Id of parent circle
Author Id	Group Id of author identity
Description text	Arbitrary string
Authentication policy	32-bits flags
Distribution control flags	8-bits flags
Admin key	2048-bits RSA **public** key
Publish key [optional]	2048-bits RSA **public** key

Messages are hierarchically organised data items that belong to a given GXS *group* of a specific *service*. They provide both parenting and versioning information in the form of pointers to other *messages*. Depending on the *service* and *group* policies, *messages* may have multiple signatures attached.

Message Meta Data	
Field	**Type**
Message Id	128 bits hash (meta data + private data)
Group Id	Id of the parent group
Publish time	32-bits integer
Parent Msg Id	Id of parent message
Orig Msg Id	Id of previous version of message
Author Id	Group Id of author identity

Both *groups* and *messages* have their own Meta Data, which can be updated during their life-time. Group and message meta data are described in the tables below:

In addition Groups and messages have their own data, that is entirely defined by the client *service*.

Applications may use various combinations of these components defining their own authentication/signature policies at the application level or at the group level, in order to define specific distribution strategies.

4.2 Data authentication

Authentication is provided by GXS using RSA signatures. Every node in the network participates to the validation of these signatures so that a malicious node cannot corrupt the data beyond his direct friends. Three kind of keys are involved in group/message signatures for different usage, as summarised in this table:

	Group Meta Data + Group Data	Message Meta Data + Message Data
Admin signature	Required	N/A
Publish signature	N/A	Optional (service-dependent)
Author signature	Optional (service-dependent)	Optional (service-dependent)

Group signatures GXS groups are distributed with 2 different signatures: the admin signature, and optionally the author signature. The admin signature controls the content of both the group meta data and the group data. The author signature ties the group content to a specific identity. If the identity is also signed by a given node of the network (See Section 5), the chain of signatures authenticates the GXS group to that network node.

The group admin key is distributed as part of the group metadata. Its private key counterpart is kept secret by the node that acts as a group administrator.

Only the admin signature is mandatory for groups. The author signature may be left blank, if the service requires the group data to remain strictly anonymous. If not, the author signature can e.g. be used to provide a contact author to the group, which can be modified anytime by the group administrator.

Message signatures GXS messages optionnally have two signatures: the message author signature, and the group publish signature. The message author signature ties the message to a specific GXS identity (which is not necessarily the one used in the group author signature).

If required, the message publish signature authenticates that the message authors has been granted the rights to do so. While the public publish key is always distributed as part of the group meta data, the private publish key is manually echanged between nodes. This allows to e.g. grant a specific list of neighbor nodes publishing rights to a given GXS group.

Figure 2 summarizes signature validation and data interdependencies between GXS groups and messages. Services can benefit from the different combinations of group and message signatures, in order to reach various levels of authentication or anonymity. This is was we call the authentication policy, which we describe later on in Section 4.5.

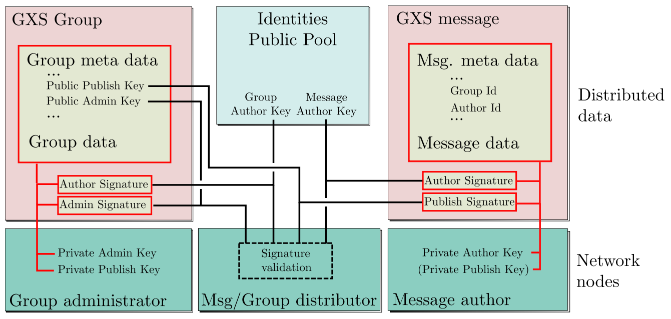


Figure 2: Signature and validation for data distribution in GXS. The bottom row represents the respective jobs of group administrators, message authors and distributors which are any node in the network having visibility to the GXS group, and who will collaborate in validating the data.

4.3 Distribution

All GXS data is synchronised using a cascaded algorithm that is designed to limit the network load as much as possible: except for messsage IDs, the data that is transfered between nodes is restricted to the data that the node does not have yet.

In order to achieve this, a peer broadcasts synchronization requests to its neighbours every minute, for known groups and known messages (See Figure 3 and 4). Each request respectively contains the last known time stamps for the set of group meta data (resp. group messages of a given group), in a format that does not allow the server to know which groups are requested unless the server itself is subscribed to the group. The rest of the traffic only depends on the need to actually perform a data transaction, which depends on a seried of tests.

Importantly, the algorithm always compares time stamps that belong to the same machine. This is achieved by storing locally the time stamps previously provided by servers into a "client update map". In each synchronization request, servers receive their own timestamp as known by the client which they previously provided, and compare it to the modification time of the local database.

In order to improve robustness, transactions are handled in multi-part format when needed. This is especially required for message data, which size limit may depend on the specific application. Message lists are also limited in size, and may require multiple transactions to be entirely transmitted. In addition, clients may choose to only request the most recent messages sorted by their publish time stamps, which also limits the traffic when a new node needs to sync all existing messages and groups at once. Applications may also choose a specific maximum age of message synchronization and storage.

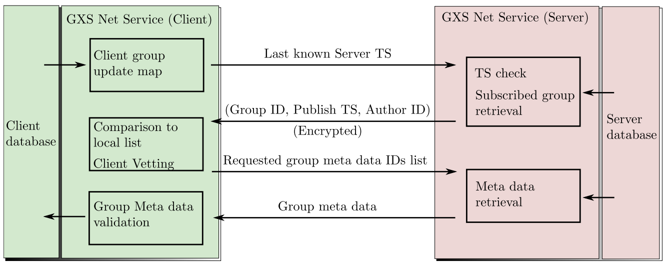


Figure 3: GXS Group synchronization logic. Clients broadcast the last known time stamp for group meta data to each connected node, triggering a cascaded synchronisation of information where only new or updated data is transferred.

As illustrated on Figures 3 and 4, group meta data and message data that is received is validated w.r.t. the required signatures, before to be stored into the database.

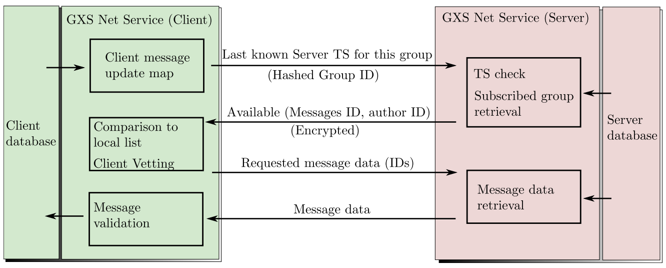


Figure 4: GXS Message synchronization logic. Similar to groups, a cascaded synchronisation only exchange new or updated message data.

4.4 Privacy

The distribution of GXS groups and messages is optionally conditioned by circles of identities, with the following model: a group restricted to a circle should only be known and accessible to the members of this circle. To achieve this, group and message data distribution is partially encrypted (See Figures 3 and 4):

- Group IDs and message IDs of restricted groups are sent encrypted using envelop encryption (RSA+AES), for all members of the circle the group is restricted to.
- Message synchronisation requests for restricted group are sent hashed according to

$$\mathrm{ID}_{hashed} = \mathrm{SHA1}(\text{Group ID}|\text{Destination node ID})$$

Envelop encryption is handled transparently by the GXS network service which attempts to decrypt received encrypted items using the private RSA keys of the locally hosted identities. If successful, the data is passed to the synchronisation routine.

The key IDs used for envelop encryption are not included in the encrypted data (only the 256-bits encrypted versions of the common AES encryption key are), so that the list of members of the group cannot be obtained from an encrypted group synchronisation packet. Hiding the number of group members is also possible, by adding an arbitrary number of fake encrypted AES keys.

Although message synchronisation requests could be sent encrypted, the cost of envelop encryption/decryption can be rather high. For this reason we prefer to send hashed group IDs salted with the target node ID. Doing so the request cannot be returned to the sending node for recovering the information. Conversely, if the destination has been granted visibility for the GXS group it will be easy to bruteforce the group ID for known groups and keep the result in a cache.

Finally the list of available messages for restricted groups is sent encrypted, in order to make sure that a previous circle member who was later denied from that circle cannot read the new message IDs. Conversely, a new member of the group restriction circle will have access to all existing message in that GXS group.

4.5 Services

The service layer of GXS define software components that automatically benefit from the GXS data distribution system, provided that they define the data to be distributed and properly talk to the GXS backend. In practice, creating a new service to distribute data requires very little effort: The new service indeed only needs to provide three pieces of information:

1. the definition of its own data to be shared between nodes and how to serialise it;
2. the authentication policy to be used;
3. a set of interface methods that convert service information into GXS data, for instance user interactions and notifications.

Service data. Each service needs to define the data that will be shared. This is achieved by deriving abstract `GxsGroup` and `GxsMessage` classes which already contain the GXS meta data needed for distribution (See Section 4.1), and adding additional data required by the new service. A Forums service will for instance need to derive its own `GXSForumGroup` class to add forum name and description, and its own `GxsForumMessage` to store threaded messages. Services also need to provide a serialiser for each new data type.

Authentication and subscription policies. In order to define its access and publishing models, a GXS service provides flags to require if applicable, the group author/publish signature and the message author/publish signatures.

For more flexibility, message come in two types: root messages and child messages. The Channels service for instance, allows a single anonymous person (the channel publisher) to post data items, each creating a new thread in the channel. Subscribers to this channel however can post a response to each post which appears in the service UI as a comment. As such, channels require no author signature for groups, but only the mandatory admin signature, require a publish signature and no author signature for root messages, and author signature only for child messages.

In addition, each GXS group of a given service may have its own distribution policy: it can be either public, or restricted to a given GXS circle of identities.

This complete set of rules offers a great deal of possibilities to define new services.

Service interface. Services need to collect notifications from the GXS transport and authentication layer that new GXS groups and GXS messages are received, and grab the service private information from them, most of the time in order to display it in a graphical user interface.

On the other way around, services collect user-created data, convert it into their own GXS-derived data and call the GXS group/messages publishing methods.

Practical examples. At the time of writing, multiple GXS services have been implemented, and cover a large span of possibilities:

- Internally, the GXS distribution layer depends on two services: Identities (Section 5) and Circles (Section 6), which are used to distribute signature keys to authenticate users and sets of users to which distribution of GXS groups should be limited.
- Forums and channels correspond to a hierarchical distributed publication system (which graphically look like usenet forums and streaming channels). They are described in Section 7.

Figure 5 summarizes the authentication and data components of these four services, each of which only represent a few hundreds lines of code, in addition to the common GXS backend. In section 9, we will discuss additional possible use cases: website distribution, decentralised git repository, messaging, and decentralised blogging.

	Identities	**Circles**	**Forums**	**Channels**
Distribution	On request	Automatic	Automatic	Automatic
Subscription	Never	Automatic	Manual	Manual
Group data	Name, avatar,...	Name, Invitee list	Forum info	Channel info
Message data	[None]	Subscribe req.	Forum posts	Channel post/comment
Group signatures	Admin	Admin	Admin	Admin
Root signatures	[None]	Author	Author	Publish
Child signatures	[None]	Author	Author	Author

Figure 5: Distribution, authentication policies and private data associated with four different instances of a GXS service, showing the large flexibility of the system.

4.6 Database

All GXS data is stored in an encrypted database. In our implementation we use the `SqlCipher` library to store these databases on the disk. The cost of retrieving and storing information can be rather significant and in order to avoid slowing down the client software, all accesses to the database are made asynchronous, using a token system. The key to the database is stored encrypted on the disk, using the PGP key of the node having access to it.

5 Identities

Pseudo-anonymous identities are used by GXS to distribute authored data with cryptographic guarranties. They are used as signing author to group messages, contact autor for specific GXS groups, and can also be grouped into circles in order to give limits to the distribution of other GXS groups (See Section 6). Identities are distributed in the network so as to follow the GXS groups that mention them.

Identities come in two brands: (1) Signed identities signed by the PGP key of the node they belong to. Signed identities therefore benefit from the web of trust that comes with PGP signature validation. (2) Pseudo-anonymous identities are strictly anonymous.

Identities in GXS are represented as individual GXS groups. The group data for an identity contains the following information, all of which are optional:

- the identity avatar, in PNG format.
- the PGP signature of the node in the darknet that holds that identity

The name of an identity is given by the description text of the group meta data.

Users can create an arbitrary number of them, which may be the source of uncontrolled behaviors in the network, thereby motivating the need for a reputation-based control.

5.1 Distribution

Identities are distributed on demand. Everytime an identity appears in a GXS message signature validation, or appears as a group contact author, it is requested to the network node who passed the group information for the last time, which somewhat guaranties that the node will already have the corresponding identity group.

Identities are automatically deleted when unused. For that, the identity service keeps a time stamp about the last time usage of GXS identities accross all services. This is easily achieved by recording every time an identity is needed to validate some data. If for some reason an identity is deleted while still needed, the system will however request it again to neighbor network nodes.

Because identities are GXS groups, they benefit from all the group distribution control. Although not currently used in our implementation, it is still possible to limit the distribution of an identity to a specific GXS circle, making that identity otherwise invisible.

5.2 Reputation based control

Every social network has its trolls, and that certainly includes darknets where people can reach an unprecedented level of anonymity. Identities in GXS are therefore also used in conjunction with a reputation level system to allow users to locally control the distribution of data.

The role of the distribution control system is to prevent disturbing data to spread without control, while not sensoring new users. In particular:

- data that is posted by a new author should have sufficient visibility to bootstrap the voting process
- the user should keep full control over which posts he will or not forward to his/her friends
- users who do not supply an opinion about an identity should be able to rely on the overall opinion at neighbor nodes
- the system should by design prevent badly intentionned users to create new identities to escape reputation control

To do so, every user in the network is given the opportunity to supply an opinion about each identity he sees, which is mostly driven by his assessment of how the identity behaves in the network, for instance when posting offending messages. An opinion is either *positive*, *negative*, or *neutral* which is the default. Opinions are shared between neighbor nodes only (see the discussion at the end of this section). Given its own opinion and the opinion of friend nodes, each node determines a local reputation level for each identity:

Own opinion	Total votes at friend nodes	Reputation
Negative		Negative
Neutral	$V_- > V_+$	Remotely negative
Neutral	$V_- = V_+$	Neutral
Neutral	$V_- < V_+$	Remotely positive
Positive		Positive

The distribution logic is the following: a node always receives data, but only distributes it if the reputation of the data author allows it, depending on 3 factors: (1) the reputation of the author, (2) the anti-spam level of the GXS group considered and (3) the level of trust the node can have in that identity. This is illustrated in Figure 6.

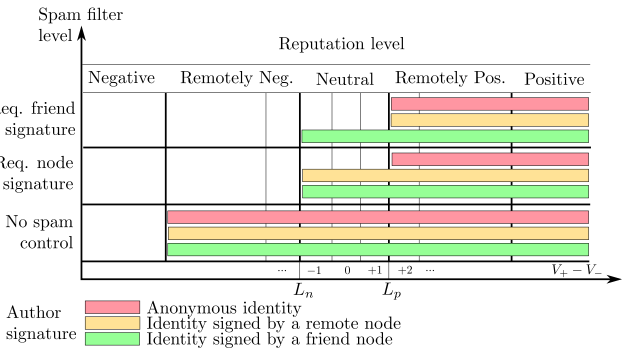


Figure 6: Minimum reputation required to forward GXS messages and groups for 3 different levels of spam control. The threshold changes based on friendship relationships between nodes in the darknet: anonymous identities are less trusted than signed identities by unknown nodes, which are in turn less trusted than identities signed by known nodes, a known node being a friend node of parametrizable distance.

GXS Groups are equiped with distribution control flags (in the group meta data, see Section 4.1), which request more or less strict reputation control over the users. In the most permissive setting (See Figure 6) all posts are forwarded, except for the identities that are locally banned by the node. In the most strict setting, posts signed by identities that are either anonymous or signed by a non friend node require a strictly positive reputation in order to be forwarded to friend nodes.

Distribution control flags are controlled by the administrator of the group. Each user is of course technically allowed to not follow these rules, but his behavior will not affect nodes beyond his direct friends nodes.

Note: We have of course considered the alternate possibility to represent up/down votes using GXS messages in the corresponding identity group, since GXS would by design provide distribution and authenticity of these votes. However, in this case, only a careful examination of the graph of signatures would allow to detect forged reputations [ACBM08]. This operation would also be limited to trusted network nodes, which in the case of a darknet boils down to connected neighbor nodes. We also prefered to have the distribution locally adapt to reputation rather than globally, since the network connexions are based on friendship—and are likely to stand for a common interest—and hardly change during a network's life time. Our local distribution control system also has two practical advantages:

- it prevents anyone from purposedly modifying the reputation of an identity beyond his direct friend nodes;
- to hide the opinion of nodes over identities beyond their direct friend nodes. This avoids troll wars in practice.

6 Circles

Circles represent sets of identities that can be used to restrict the distribution of other GXS groups in arbitrary GXS services. For instance a specific forum may be restricted to the members of a circle. In such a case only the members of the circle can see that the restricted group actually exists, and access its content, the restriction being based on the encryption of the data when passed to friend nodes (See Section 4.4).

Circles are distributed among the network and follow the GXS groups of which they restrict the visibility. In addition to ensuring the privacy of data that is retricted to some circles, the design needs to meet two important security requirements:

1. circles membership should depend on an administrator who can invite people to be in the circle, and revoke its members;
2. people should be able to request membership without automatically be true members of the circle. They should also be able to get out of a circle without permission of the administrator.

These two conditions effectively prevent someone with bad intentions to create a circle with non solicitated members, and allow people to exit a circle whenever they want to.

Circles are implemented as a GXS service, where each circle is a GXS *group*. The group data contains the list of identities that are invited by the administrator. Membership requests are posted in the circle as *signed group messages*. For each identity, only the last message is relevant. It can be a positive membership request (if the identity wants to request membership) or a negative request if that identity wants to quit the circle. Each peer in the network performs the following tasks:

1. for each GXS id who posted messages in the circle group, only keep the latest subscription request, and intepret it as the GXS id willing to be or not in the circle

2. intersect the list of positive subscription requests with the invited list, in order to form the list of members in that circle

Since the invited list is signed by the administrator (the creator of the circle), no one can hack himself into a circle member nor remove other members. Although it is possible to locally delete membership requests from other identities, it is unlikely that other nodes in the network will not get these group messages from a different path.

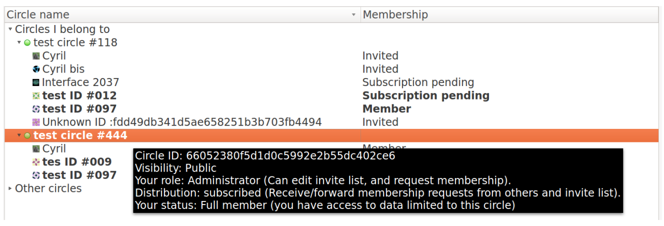
In addition, the circle administrator is given the possibility to add members (as a response to a pending subscription request), and an invited identity may join the circle if invited or simply request membership if not. The GXS distribution system automatically takes care of exchanging these data (group data and group messages) among peers in the network.

In this system, everyone in the network automatically checks that someone in a circle has actually requested to be in that circle or that he is part of the invited list, and therefore globally participate in the authenticity of the data.

Private and self-restricted circles As circles are GXS groups, they can be restricted to an other circle. That includes the possibility to restrict a circle to itself. Members of a circle restricted to another circle only see that circle if they are explictly allowed to. In the specific case of a self-restricted circle, only the invited people in the circles will be able to see that the circle exist.

Subscription policy Subscription to circle *groups* are handled automatically so that invited members see the circles they are invited to, and identities who request subscription forward these subscriptions back to the administrator.

An example of circle data with pending and accepted membership is displayed in Figure 7.



Circle name	Membership
Circles I belong to	
test circle #118	
Cyril	Invited
Cyril bis	Invited
Interface 2037	Subscription pending
test ID #012	**Subscription pending**
test ID #097	**Member**
Unknown ID :fdd49db341d5ae658251b3b703fb4494	Invited
test circle #444	
Cyril	Member
tes ID #009	
test ID #097	
Other circles	

Circle ID: 66052380f5d1d0c5992e2b55dc402ce6
Visibility: Public
Your role: Administrator (Can edit invite list, and request membership).
Distribution: subscribed (Receive/forward membership requests from others and invite list).
Your status: Full member (you have access to data limited to this circle)

Figure 7: Graphic display of the circle administration tool in the *Retroshare* software. Bold generally means that the user owns the data (bold identities are self hosted, bold circles are administrated locally. The tooltip indicates the distribution and administrative rights for the second circle "test circle #444".

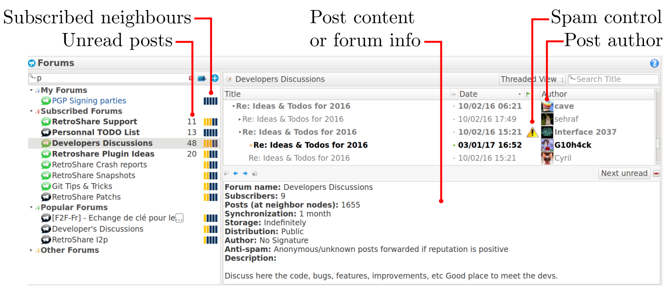
7 Forums and Channels

Forums and channels are user-based threaded data sharing applications that both leverage the generic nature of GXS. They are presented together to illustrate the possibilities offered by the system in terms of content management and representation.

While forums can be written by all users, channels are read-only and require the private publish key to post or edit a message. Channels however allow users to send comments and also vote them up and down. They are displayed as a threaded hierarchy of posts (See Figure 10). The GXS representation of posts, votes and comments is summarized in the table below:

	GXS Primitive	Write access
Channel topic	Group	Group administrator
Channel post/edit	Message subclass	Group publisher / post author
Channel comment	Message subclass	All users
Channel vote	Message subclass	All users
Forum topic	Group	Group administrator
Forum post/edit	Message subclass	All users / post author

Channels and forums allow users to edit their posts. This is made very easy by the GXS system by simply re-posting the message with the previous message version set as original message. At the time of display, channels only display the latest version of each post, while forums allow users to see all previous versions interactively.



My Forums
PGP Signing parties
Subscribed Forums
RetroShare Support 11
Personnal TODO List 13
Developers Discussions 48
Retroshare Plugin Ideas 20
RetroShare Crash reports
RetroShare Snapshots
Git Tips & Tricks
RetroShare Patchs
Popular Forums
[F2F-Fr] - Echange de clé pour le...
Developer's Discussions
RetroShare I2p
Other Forums

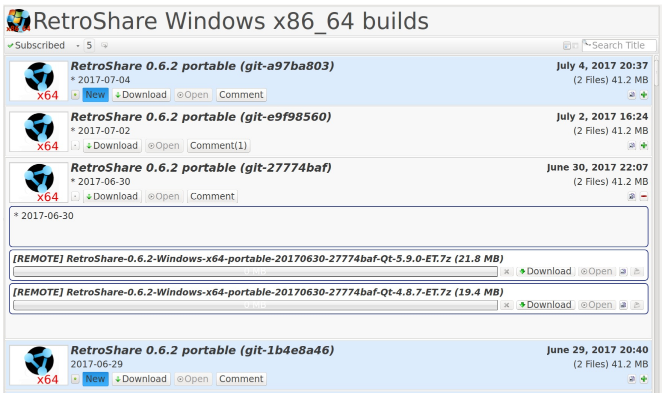
Title	Date	Author
Re: Ideas & Todos for 2016	**10/02/16 06:21**	**cave**
Re: Ideas & Todos for 2016	10/02/16 17:49	sehraf
Re: Ideas & Todos for 2016	**10/02/16 15:21**	**Interface 2037**
Re: Ideas & Todos for 2016	**03/01/17 16:52**	**G10h4ck**
Re: Ideas & Todos for 2016	10/02/16 15:21	Cyril

Forum name: Developers Discussions
Subscribers: 9
Posts (at neighbor nodes): 1655
Synchronization: 1 month
Storage: Indefinitely
Distribution: Public
Author: No Signature
Anti-spam: Anonymous/unknown posts forwarded if reputation is positive
Description:

Discuss here the code, bugs, features, improvements, etc Good place to meet the devs.

Figure 8: General view of forums. A list of forum topics is shown on the left, each of which is internally a GXS group. The threads of posts for the selected forum topic is displayed at the top right and information about the forum is shown below. If a post is selected this information is replaced by the content of the post.

The various types of messages (post, comment, vote) are all created by deriving the base GXS message class into a specific message class. Most of the code needed by the application layer that implements channels and forums is the graphical user interface. The core part only needs the definition of the


RetroShare Windows x86_64 builds

Subscribed 5 Search Title

RetroShare 0.6.2 portable (git-a97ba803) — July 4, 2017 20:37
* 2017-07-04 — (2 Files) 41.2 MB
New Download Open Comment

RetroShare 0.6.2 portable (git-e9f98560) — July 2, 2017 16:24
* 2017-07-02 — (2 Files) 41.2 MB
Download Open Comment(1)

RetroShare 0.6.2 portable (git-27774baf) — June 30, 2017 22:07
* 2017-06-30 — (2 Files) 41.2 MB
Download Open Comment

* 2017-06-30

[REMOTE] RetroShare-0.6.2-Windows-x64-portable-20170630-27774baf-Qt-5.9.0-ET.7z (21.8 MB) — 0 MB — Download Open

[REMOTE] RetroShare-0.6.2-Windows-x64-portable-20170630-27774baf-Qt-4.8.7-ET.7z (19.4 MB) — 0 MB — Download Open

RetroShare 0.6.2 portable (git-1b4e8a46) — June 29, 2017 20:40
2017-06-29 — (2 Files) 41.2 MB
New Download Open Comment

Figure 9: List of posts in a channel. Each post may have comments (shown on Figure 10), an attached image, and references to files that can be downloaded by the user.

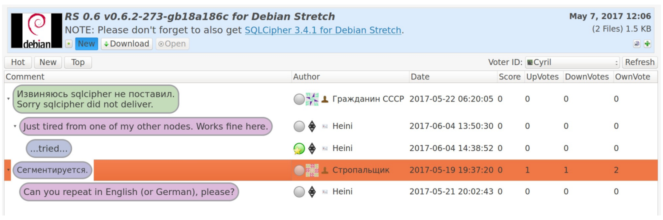
message types, the distribution policy flags (See Section 4.3) and a few interface functions to convert internal data types into GUI-compatible data, for a total of less than 300 lines of C++ each.

Although not implemented yet, it is straightforward to set additional user priviledge to some GXS identities, wich can therefore be enforced by all participants in the network. Having forum moderators for instance simply requires:

- the forum admin to add in forum group data the list of moderators (this information is automatically signed by the administrator)
- implement e.g. post deletion as a dedicated group message signed by one of the moderators in the list
- every user in the network will be able to enforce the rules: not display local deleted messages and not forward them to friend nodes.
- this system obviously provides a possibility to cancel a moderator's work if the administrator removes the moderator from the signed list.

7.1 Secure transfer of attached files

Forums and channels may contain references to potentially large shared files, which should be possible for a distant node—having access to the channel—to download, without the need for relay nodes in the channel/forum to actually store the file. This is made possible by a tunneling system that guarranties the anonymity of the source and destination, very similar to the *Turtle* file transfer protocol [PCT04]. Our own implementation allows swarming, and doesn't use global addressing.


RS 0.6 v0.6.2-273-gb18a186c for Debian Stretch — May 7, 2017 12:06
NOTE: Please don't forget to also get SQLCipher 3.4.1 for Debian Stretch. (2 Files) 1.5 KB
New Download Open

Hot New Top — Voter ID: Cyril — Refresh

Comment	Author	Date	Score	UpVotes	DownVotes	OwnVote
Извиняюсь sqlcipher не поставил. Sorry sqlcipher did not deliver.	Гражданин СССР	2017-05-22 06:20:05	0	0	0	0
Just tired from one of my other nodes. Works fine here.	Heini	2017-06-04 13:50:30	0	0	0	0
...tried...	Heini	2017-06-04 14:38:52	0	0	0	0
Сегментируется.	Стропальщик	2017-05-19 19:37:20	0	1	1	2
Can you repeat in English (or German), please?	Heini	2017-05-21 20:02:43	0	0	0	0

Figure 10: Comments thread in a channel. Comments can be rated and sorted using a voting system that is also synchronized by GXS.

If the forum/channel is private, the file transfer should however prevent relay nodes along a tunnel to evesdrop on the transferred data, or even to know what file the data belongs to. Transferring files while guarrantying both strict anonymity and end-to-end encryption robust to man-in-the middle attacks obviously is a chicken-and-egg problem since it would require a pre-shared a secret key between the two ends of the tunnel that want to stay anonymous. Let's call $H()$ a cryptographic hashing function. We solve the problem in a way similar to what file transfer on GnuNet does, without the use of temporary authentication keys [], in order to guarranty the strict anonymity of the source and destination of the tunnel:

1. Tunnels are requested by broadcasting requests for $H(H(F))$. This way only the source of the file—who knows $H(F)$—can answer the tunnel request;
2. data in every packet along the tunnel is encrypted (with Chacha20) using a symmetric pseudo-random key that is derived by mixing a cryptographically random $96-$bits initialization vector with the 32 bits tunnel ID and the actual hash of the file $H(F)$;
3. each packet is authenticated using HMAC based on that random key.

This construction follows the AEAD that is described in the Chacha20 cipher suite [NL15]. If the hash is published in a private group, only members of that group will be able to access the file using anonymous tunnels. Users who also have that file will be able to swarm it, without being aware that this file is shared in a particular GXS group.

8 Asynchronous messaging

Sending/receiving e-mails in a totally decentralized system is very challenging: a reliable decentralized email system needs to provide transport, authentication, integrity, and confidentiality in an asynchronous way. Peers may not be simultaneously online while the message is transmitted. The GXS plateform offers a straightforward way to implement such a messaging service, based on the underlying transport paradigm, and provides the following guarranties:

- messages are encrypted and signed
- message recipients are hidden
- transport is guarrantied by an anonymous return receipt system

Transport

A limited set of GXS groups is created on demand to store the pending e-mails for all recipients. These Email Transport Group—referred to as ETG—have no publish control (*i.e.* no publish key signature is required). E-mails are transported as GXS messages in these groups, and are therefore authenticated by the public key of their authors, quite like forum posts. E-mail transport groups are automatically created and deleted according to the following set of rules:

1. messages are always posted into the group with the oldest creation time stamp, with less than a maximum number of messages
2. a new ETG is automaticaly created when no ETG with free space is available
3. nodes automatically subscribe available ETGs that provide messages (and therefore automatically advertise them to their neighbor nodes)
4. messages in ETGs that are older than the maximum message transport duration are deleted
5. ETG with no messages are deleted

Rules 1, 2 and 3 ensure that the number of groups that is used is as limited as possible. It will increate if a lot of messages need to be transported, and decrease automatically after old messages are delete. It is quite likely that a node in the darknet which sends a message without having access to any group will create a new one, but when he connects to other nodes, group selection will consistently favor one of the two groups (Rule 1) and prevent the other one to be spread.

Message protocol

The e-mail transport is all based on publishing appropriate GXS messages in the e-mail groups:

When Alice wants to send an email to 'Bob', an identity picked from the public key pool (See Figure 3), Alice forms a data packet containing three parts:

1. a random message Id
2. the email itself, encrypted for Bob
3. a return receipt signed by Alice containing the above Id, and encrypted for Bob

The packet is serialised and posted as a GXS message in of the ETGs available (openning a new thread in the group). On receipt, Bob decrypts the email message data, sends it the UI for display, decrypts the return receipt signed by Alice and posts it as a GXS message in the same thread.

The GXS transport layer automatically distributes the message and return receipt to all connected peers in the network, each participating as a cache for the messages. Network nodes will therefore perform the following tasks:

- a decryption attempt is performed on newly received GXS messages in subscribed ETGs. If successful, the GXS message is handled as a received email
- messages and receipts older than the email storage period are deleted

The fact that GXS checks and requires post signatures in the group messages that are involved in option 2 prevents any man-in-the-middle attack of the messaging protocol.

For efficiency reasons, messages are only stored in the GXS emailing group for a short amount of time, in order to limit the amount of data that the emailing group is handling while still providing as smooth asynchronous transport as possible. The emailing client application is responsible to re-post any un-acknowledged message beyond the TTL of that message is reached. As a destination, it also takes care of duplicate emails that can be produced by this double posting.

Because the authentication policy of the GXS messaging service requires signature of GXS messages, the sender is known, but the recipient of each message is unknown. This way, the integrated anti-spam prevents someone from flooding the service with fake messages. Hiding the destination Id of messages forces all nodes to attempt the decription of messages using all available private keys, in order to figure out whether they are the true destination of the message.

Message encryption

Multiple options are available for encryption (Currently only the first one is implemented in our system):

1. Messages are encrypted with RSA using the public key of the e-mail destination, using anonymous envelop encryption.
2. Messages are encrypted using ephemeral keys. In order to do this, the destination needs to generate and post a DH ephemeral key $g^a[\text{mod } p]$ in the messaging group in advance for future e-mails. The email client will pick this ephemeral key, generate a random one-time ephemeral key $g^b[\text{mod } p]$ and post the encrypted message m as $(g^b[\text{mod } p], \text{AES}(g^{ab}[\text{mod } p], m))$

Option 2 is a littlemore constraining since every peer needs to publish at least one ephemeral key in advance in order to be contacted. This scheme however provides forward secrecy by design, which is important in this case since no control is given over who accesses the encrypted messages. Note that this protocol cannot prevent two users from incidently using the same server's ephemeral key twice, because of which p should be chosen to be a safe prime [MU10]. Both methods allow a message to be encrypted for multiple users at once, using encapsulated public key encryption, while not requesting to disclose the identity of the recipient.

9 Discussion of possible use cases

In this section we discuss the possibility to implement additional services, which mostly require an effort in graphical user-interface deployment and the representation of internal application items using GXS primitives. The GXS engine will automatically handle the data distribution, consistency, authentication and privacy aspects. We strongly encourage—and offer our help—to readers who want to implement these applications.

Distributed blogging and micro-blogging would be a fully decentralized clone of twitter (or any blog system). It can be implemented in a way very similar to channels, where only the publisher can post messages. Users can post comments, vote over posts, etc. Message data may include images and links in their private part.

Directory synchronisation allows to synchronize folders accross computers. The private group data is used to store the folder content. Synchronisation is locally handled by the service part, checking which files are already here or not, possibly comparing their hash. The synchronization can happen on multiple computers, all having the same reference directory to synchronize, but globally swarming to sync the files. The access to the directory content can be made private by restricting the sync group to a circle.

Distributed calendar and tasks. Each calendar is a GXS group where write access is limited to the list of publishers, and read access to the content of the calendar can be limited by restricting it to a circle. Tasks and events are messages, each containing dates and status as private message data. Sub-tasks and comments to events can be represented as GXS messages with a different write access hence allowing users to collaborate on tasks and comment on events.

Distributed source versioning would allow to fully decentralize and secure the use of git [SHS$^+$15]. In this model, a GXS group represents a versionned project, similarly to what a git repository clone would do. Users who subscribe to the group have the possibility to clone the source for that project, which they do using the private tunnelled file transfer described in Section 7.1. They can also request to merge their own clone of the repository with the master repository through a merge request mechanism. Merge requests are represented as GXS messages, and are handled by the master node who has the administrator rights over the group, so as to centralize conflict resolution, and ensure a consistent state of all clones. Messages are also used to send confirmation of merge, and notify of conflicts if any. Normally most conflicts should be visible locally to each node that clones the repository, but it may happen that multiple incompatible merge requests are sent simultaneously by different nodes.

We believe that this implementation of the git protocol over GXS allows a fully decentralized source management system. If the master repository is hosted on a Tor-node (See Section 12), then the source project is automatically out of reach to a global observer, and still benefits from the strong levels of authentication and privacy provided by the GXS system.

Fully decentralized social network Multiple partly decentralized social networks exist already [PFS14, ZI16]. Building a fully decentralized social network on top of GXS is a simple task, although a lot of GUI work would be needed to display the information in a user-friendly way. Internally, this goal can be achieved as follows:

The home page of each user is a GXS group acting as 'master group'. GXS distributes user's posts as sub-groups of the master group. Similarly to messages, GXS groups can indeed be linked hierarchically using the ID of their parent group. GXS messages are used to represent comments, user replies to posts and votes. Because posts are themselves groups, circles can be used to provide visibility control of each separate thread of posts which automatically applies to all answers and comments to the same thread.

Such a system would benefit from other built-in GXS services: The subscription system allows to automatically promote popular pages; users would be allowed to exchange messages using the existing mail system (See Section 8); users would have the possibility to privately share files that can be downloaded privately using the tunneled file transfer described in Section 7.1.

Here again, coupling such an service to an application that runs GXS over Tor nodes would create an extremely robust and private fully decentralized social network.

The table below summarize which GXS primitives should be used to build these applications:

	GXS groups	GXS messages	W/access
Blogging	Blog	Blog posts, comments	Publisher
Directory sync	Directory content	[Unused]	[Unused]
Calendar	Calendar	Tasks, Events	Publisher
Source versioning	Git repository	Merge requests	Everyone
Social network	User page, Post threads	User posts	Publisher
		comments/votes	Everyone

10 Security

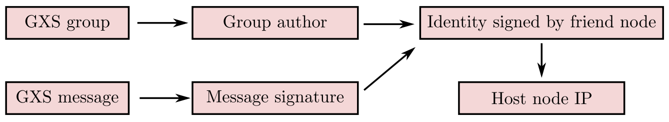
Threat model The GXS data exchange system guarranties confidentiality in a friend-to-friend mesh network where links between friend nodes are authenticated and encrypted. In this context, privacy with respect to outside attackers mostly relies on the security of TLS links between nodes, which we will not discuss here. We assume that TLS links are authenticated with PGP keys that users exchange in order to connect, which prevents any man-in-the-middle attackers from outside the network to evesdrop on the transferred data by relaying the unencrypted traffic.

Friend-to-friend networks (or Darknets) are generally vulnerable to network mapping techniques, possibly involving measurements of the amount of—encrypted—data that transfers between nodes. Using a secure DHT in the host mesh application largely mitigates this problem [CDG$^+$02]. In our implementation (The Retroshare software), a set of techniques have been implemented and successfully prevent traffic forwarding due to sybil attacks in the DHT.

In this paper we focus our security analysis on the threats caused by an attacker from inside the network. In this scenario, the attacker can read and modify the GXS code, or the code of the hosting friend-to-friend application, in every way possible. He can own his own network node(s) and participate in the network. Our analysis of security is focused on attacks against privacy, anonymity and stability of the data distribution system.

10.1 Anonymity and authentication

A loss of anonymity happens when an attacker manages to connect a supposedly anonymous piece of data with the IP address of the person who posted it. The possible paths to recover the IP of an author of a GXS message or a GXS group are depicted in the graph below:



In this graph, most nodes correspond to optional features that depend on the particular settings of each service. A forum for instance (See Section 7) may not require GXS identities to be signed nor provide a group author. A channel will only contain messages that are unsigned therefore not linking the post to anyone. Finally, note that the last link requires the attacker to have the host node as a direct friend, which in practice drastically reduces the scope of an attacker.

An attacker inside the network can of course try to perform timing statistics in order to figure out whether an anonymous GXS identity is hosted by a friend node or not. We have not implemented counter measures against this (such as imposing a time bias in the distribution of messages), since the static nature of friend-to-friend networks makes it impossible for an attacker to crawl the network in the hope of connecting to random nodes. In practice, it is safe to assume that complete anonymity is only guarantied beyond direct friend nodes.

The network layer should also make sure that it is not possible to find the IP of a node from a node signature. In our implementation, we ensure that property by basing DHT requests on the hash of SSL certificates that are signed by the node's PGP key, but only distributed to friend nodes. It is therefore not possible to request for a particular node's IP only knowing its PGP key signature.

Conversely, services that require messages to be signed by node-signed GXS identities provide a strong authentication of their data.

10.2 Privacy

The table below summarizes which piece of information is available to whom across a network that runs GXS:

Information	is visible to...
Hosting a signed identity	Hosts were identity is active
Hosting an anonymous identity	Friend nodes (with timing stats)
Membership to public circles	Friend of circle subscribers
Membership to private circles	Hosts of circle invitees
Subscription to public groups	Friend nodes of subscribers
Subscription to restriced groups	Hosts of circle members
Data of circle-restricted groups	Hosts of circle members

The table makes the difference between circle subscribers (nodes who subscribed the circle as a GXS group, for instance because they requested membership and want the information to get back to the circle administrator), circle invitees (which are explicitly invited by the circle administrator), and true circle members who are both invited and grant membership.

User impersonation. Impersonizing a GXS message author requires to forge a RSA signature knowing the public key of the author's GXS identity. This is known to be a computationnaly hard problem. It is of course possible for an attacker to create a GXS identity (using a different RSA key-pair) with the same name and avatar than an existing identity, and use this copy to post messages. The two identities will however have different IDs and will be handled as two different objects by the system.

Passive evesdropping over transfered files. The end-to-end encryption of files that we described in Section 7.1 allows users in the darknet to exchange these files anonymously, while keeping a full confidentiality over their content and meta-data, as soon as the file hash is kept secret. Consequently, file pointers that are shared in circle-restricted GXS groups are only accessible to the members of the circle.

Exposing members of a private circle. Private circles are a specific type of GXS group that is circle-restricted to itself (See Section 6). Its members are consequently only known to the list of invited identities. Both the existance of the circle and its content are kept private, since both are sent to friends encrypted using envelop encryption using the list of invited members of the circle. Exposing the members of a self-restricted circle needs to break the RSA/AES encryption that is used to exchange its data with other members of the circle.

Nodes that nodes that are friend to someone who hosts a self-restricted circle will therefore only see that some unknown GXS group exists at the next node, without being able to request the group data nor its messages.

Involving users in unsolicitated activity. This is a serious threat in a network where only friend nodes can really be trusted. Among possible attacks, one can imagine someone creating a circle of victim GXS identities with offending content and pretend that the victims teamed up voluntarily. One can also imagine securing an offending forum/channel to such a made-up circle, so that only the victims would see the offending forum/channel content.

The architecture of GXS prevents these by design since effective circle membership depends on two factors: to be a circle member, an identity needs to be included in the invited list managed by the circle administrator, but is also needs to have explicitly requested membership to this circle. An identity that does not meet these two conditions will not receive (nor be able to decrypt anyway) the meta data and content of groups limited to the circle.

An exception to this rules exist for private circles however, as described in Section 6, but the existance of the circle is only revealed to the invited identities, which severely limits the the threat of unwillingly involving users into a circle.

10.3 Data flow attacks

In this section we review the possible attacks to limit or flood the diffusion of data through the GXS system. The current implementation of GXS has been tested "in the wild" and therefore confronted to nasty users, who unintentionaly contributed to make the system robust against data flow attacks.

Intentional corruption/blockade of posts. An attacker inside the network can modify his own node in order to corrupt the data in existing GXS groups or messages. Since every group data (or group messages) that is received is validated w.r.t. the group admin key (and the group publish key if applicable, see Section 4.2), corrupted data is immediatly discarded on receipt at the next friend node. The corrupted information will therefore not spread at all.

Similarly, an attacker cannot impersonnate an existing group by generating a new group admin key, since the ID of the group (which is the identifier of that group in the network) is a hash of the public admin key. Doing so, an attacker would effectively create a new GXS group.

Blocking information is of course possible by locally disabling all GXS traffic. If a node acting as a bridge over two otherwise separated networks blocks GXS data, it will effectively prevent posts from one side of the network to reach the other side. This needs however a very restricted network topology and a single new connection between two nodes on each side will instantly allow the data to consistently reach every subscribed node.

Network flooding. Network flooding is a serious threat. One can imagine a GXS user who creates tons of GXS groups (for instance forums) or tons of GXS messages (for instance forum messages in a single GXS groups) possibly modifying his own node in order to automate the task. GXS is quite safe against such actions for two reasons:

Only subscribed GXS groups are made visible to friend nodes. As a consequence, locally creating millions of GXS groups at a given node will only cause trouble to the direct friends of this node, where a huge list of unsubscribed groups will be visible. It is of course possible to limit the amount of GXS groups a node can receive from his friend nodes, but we never had to come to such a limitation. Because it is fairly easy to display the amount of GXS groups advertised by each friend node, an attacker (or someone relaying his traffic) would immediately be spoted by all friend nodes and most likely disconnected from the network.

Groups that require message signature are very safe against message flooding if they enable the reputation system we describe in Section 5.2. Doing so, using the same identity to generate offending messages will rapidely result in the identity being flagged as bad, and its messages will be discarded. Creating new identities in order to post new messages will not allow the messages to be forwarded beyond friend nodes since the reputation system requires a strictly positive reputation in order to transmit the posts.

Because unsigned messages are strictly anonymous, GXS groups that would not require message signature would of course not benefit from the reputation system, and would be prone to spamming and flooding.

11 Performance

We examine in this section the bandwidth usage of the software, and discuss its robustness with respect to traditional network problems such as loss of data, clock differences between nodes and network failures.

11.1 Bandwidth

Figure 11 shows the exchange of data items that transfer up and down between two nodes when one of them performs a full synchronization of visible GXS groups. In the figure the node providing the data is subscribed to 23 GXS groups (which in this case happen to be forums) having hundreds of messages each. The color dots in the figure correspond to packets of items of similar types. What happens here is:

1. the client node broadcasts Group Sync Requests every 60 secs (item #01, at *left*);
2. the server answers with a list of group IDs with time stamp (item #02, at *right*);
3. the client asks for the metadata of unknown groups (item #02, at *left*);
4. the server sends the group data for the 23 missing groups (item #04, at *right*)

In the right side of figure 11, one can also see the server's "client" side of the synchronization algorithm, also sending group sync requests. These do not belong to the client synchronization data flow. The transaction items that are listed on both sides encapsulate items are used to split items of type #02 and #04 into smaller chunks. Some of the groups (3 in this case) are limited to a circle, so their meta data is sent encrypted, which explains the 3 encrypted data packets which encapsulate 3 of the 23 group sync items.

Figure 12 shows the data items that transfer once the client node subscribes to one of the groups containing 405 messages 21 of which fall into the user-defined synchronization period:

1. the client broadcasts message synchronization requests every 60 secs (item #10, at *left*);
2. the server responds with message sync containing the meta data for messages that fall into the synchronization period limits (item #08, at *right*);
3. the client requests for the messages (item #08, at *left*);
4. the server sends the message data (item #20, at *right*).

Similarly to group synchronization, the graph also shows part of the server's client synchronization items, and group syncronization exchanged for the the two sides.

Figure 13 shows the bandwidth profiles (up and down) over a 5 minutes window for a network node that is connected to 10 other nodes, and subscribed to 23 GXS groups (which are GXS forums in this measurements). The upward bandwidth profile clearly shows the regular 60 seconds spaced message synchronization request broadcast (item #01). The overall bandwidth consumption is minimal (a few bytes per second).

Encrypted data (items #05) correspond to message synchronisation requests for circle-restricted GXS groups (See Section 6). Individual encrypting small items using envelop encryption represents a non negligible bandwidth overhead, which make these items more visible in the graph.

The "Group stats" (item #03) not strictly necessary to the functioning of GXS, but are used to supply neighbor nodes with statistics about how many messages can be found in visible public groups, without the need to actually download them, which helps promoting active groups.

11.2 Robustness

Data redundancy. Data redundancy in GXS is extremely high: all nodes subscribed to a group host the posts for the locally chosen storage/sync periods. As a consequence, if a user loses his data, he will rapidly get it back. Besides, synchronization for a group can be provided by any connected node that is subscribed to this group. Therefore in a popular group, data always finds its way to all synchronized nodes in a short amount of time even if a large subset of the subscribers are offline.

Sharing publish keys and group administrator keys between friend nodes is an efficient way to backup the GXS groups. This should be implemented in the host application (Our implementation currently only allows to share publish keys). This mechanism can be adapted to some particular use cases, for instance by encrypting the publish keys sent to friends, and requiring a passphrase to recover them.

Time synchronisation. Time shifts between friend computers are never a problem because synchronization time stamps between different clocks are never compared to each other during synchronization as explained in Section 4.3: for every friend node, GXS always compares the time stamp given by the friend node with its own clock, to the previously recorded time stamp. When enforcing the synchronization period however, the publishing time of messages is compared to the local time on the synchronized node. The synchronization window

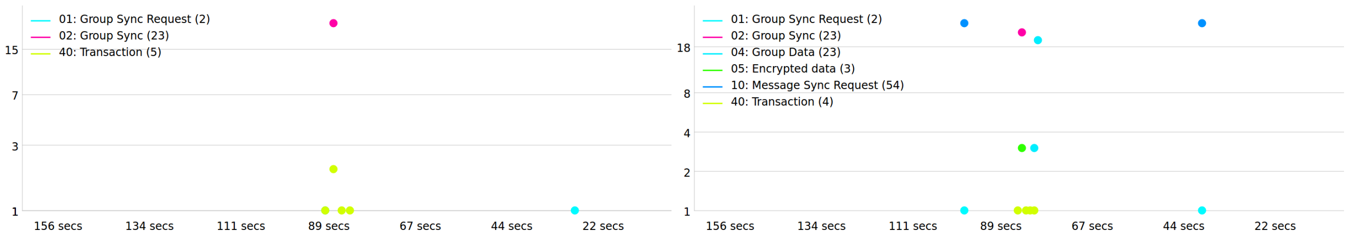


Figure 11: **Synchronization of visible GXS groups**. The dots corresponds to GXS data packets of various types up (left) and down (right) when a network node connected to a single node synchronizes the GXS groups that are reachable through this node. One can see the broadcasted group synchronization requests, followed by group data, all encapsulated into transactions items, which allow to chunk the transferred data.

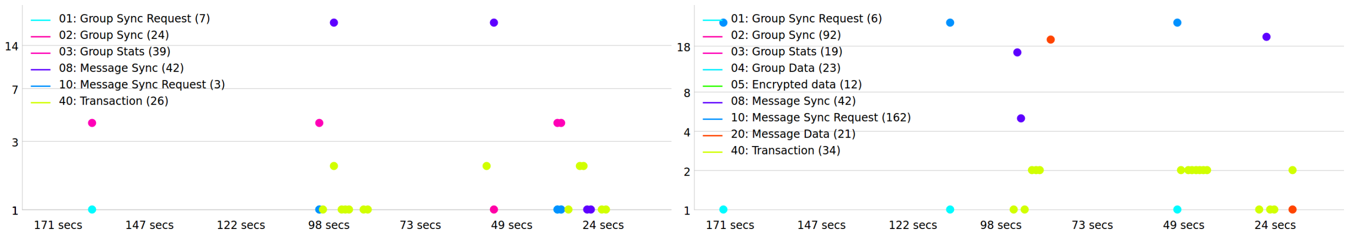


Figure 12: **Synchronization of the messages for a single newly subscribed forum**. The dots show the exchange of individual data packets that take place when synchronising a newly subscribed GXS group. See the text in for details.

(6 months by default) is therefore up to the time shift between computers, which appears to be negligible.

Time changes on a single computer do not cause any harm either, since the decision to re-synchronize groups is based on time stamps being different rather than larger. As a consequence, daylight saving time change do not have any impact on GXS synchronization.

Network failures. All GXS transactions are acknowledged and therefore can be restarted when uncomplete or cancelled, which happens when they take more than some hard-coded time limit. When a node disconnects during a GXS transaction, the transaction will be cancelled on the connected nodes and re-started at the next time of connection. In order to avoid endless loops due to very large transactions (for instance caused by someone subscribig to a group with an arbitrary large number of messages) transactions are limited in size and in the number of messages. At every synchronization the list of messages to request is computed on-the-fly so as to only request what is still missing. As a consequence, a node that disconnects very often will eventually manage to synchronize a large groups, receiving group messages by packets.

Service heterogeneity. Network nodes must be robust to unknown information coming from GXS services that are not locally supported. In our implementation this is achieved by labelling data packets according to their "service ID", and by dropping the packets that correspond to unknown services.

Limitations We identified the following limitations to the GXS system: The static decentralized nature of the network makes it prone to the creation of disconnected islands for the same group, in which case the separated parts will not synchronise until a new path is found between them. When that happens however, full synchronization is reached within minutes. The same happens when a new user joins a group with many messages.

In order to distribute messages of a given circle-restricted group, GXS implicitely supposes that at least some of the neighbor nodes host identities that belong to that circle. By construction this is always true since only subscribed groups are advertised to friend nodes, but the network topology may change with time. The information of which identity in a circles is hosted at a specific neighbor node is not required.

Anonymous identities hosted at a node lose a certain degree

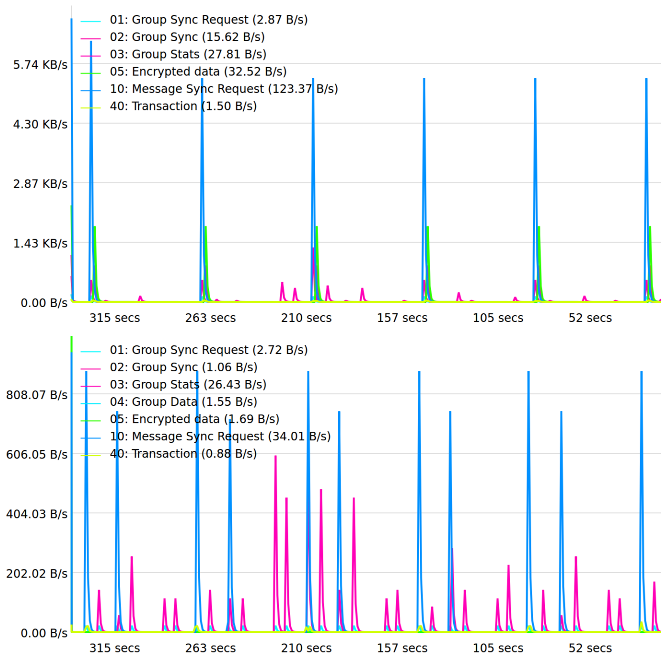


Figure 13: Linear scale bandwidth profiles accross time (up and down) over a sliding 5 mins window, for a network node connected to 10 other nodes, subscribed to 23 GXS groups, after synchronization. The numbers in parenthesis show the average bandwdith over the full measurement time period. The overall bandwidth consumption is minimal.

of anonymity w.r.t. direct neighbor nodes. It is indeed easy to compare message signature times with the time at which the data is received to figure out that the signed of the message is hosted at a neighbor node. In a F2F network, this kind of threat is not a practical concern, so we did not implement mitigation measures such as randomly removing some minutes to the time in the signature of messages.

Message synchronization is not instantaneous, and therefore can hardly be used to distribute chat messages. At the moment the polling of messages happens every 2 minutes. It can be made faster, but is already fast enough for all implemented services (e.g. forums, channels, etc).

12 Implementation

We have deployed GXS into the Retroshare application [1], in less than 20,000 lines of C++ code. Retroshare is an open-source friend-to-friend application that builds a darknet where friend nodes only communicate using trusted connections encrypted with TLS. The software is currently used by thousands of users daily [2]. The TLS links between friends can be established over TCP/UDP connections but also over Tor [DMS08] or I2P [ZH11]. Having a running implementation of GXS we had the possibility to measure its efficiency and limitations in a working context. We also faced a number of unexpected challenges which we describe now.

Network load. Our implementation of GXS does not rely on users to be online all the time. In practice, in a Friend-to-Friend network users keep disconnecting, or can disappear for a while. In these situations, as well as when a newcomer joins a part of the network that holds a lot of data, care must be taken to sync the GXS databases without overflooding the links between nodes. This happens for instance when a user subscribes a GXS group that holds many messages. To that matter, synchronization events are represented by network transactions that can be chopped into smaller pieces if required. Transactions can be cancelled when they take too long (typically when a friend node disconnects), and are restarted when the network link is up again. Our implementation also keeps lists of rejected messages (mainly due to the anti-spam system) so that these messages will not be requested again. Synchronisation of large GXS groups usually takes up to a few minutes to complete.

Database access and storage is also challenging. First of all, all GXS data that is stored on the disc—which includes private keys for GXS groups—is encrypted on the fly using AES. This is performed using a version of the sqlite library that supports on-the-fly encryption [3]. The sqlcipher encryption key is randomly chosen by each network node and stored encrypted by the TLS public key of that node. At the time of starting a node, the TLS key is decrypted using the node PGP key and used to also decrypt the sqlcipher key.

Second, because database access is not instantaneous being for decryption or even disk usage access load, all access to the GXS database are made asynchronous using a token system: retrieving the list of GXS groups for instance requires to post a request, store the token, wait for the response callback to be called and then use the token to actually access the data. While this asynchronous access system makes development a bit more complicated, it allows the user interface to re-

[1] See `http://retroshare.net`

[2] This estimation is obtained thanks to the DHT which helps users to synchronise UDP connections through NATs. People who do not use the DHT are therefore not counted.

[3] See `https://github.com/sqlcipher`

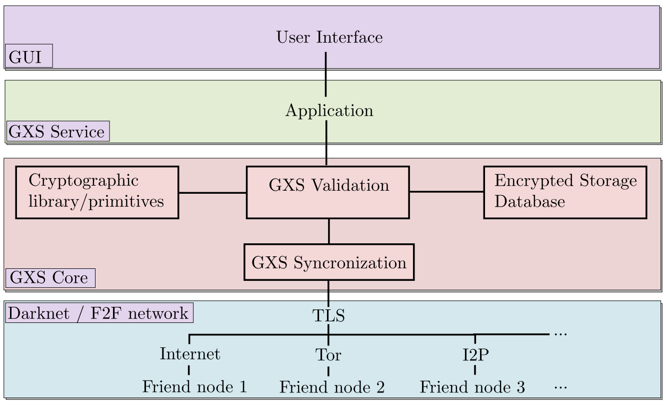


Figure 14: Overview of how the GXS components insert into a darknet application. The graphical user interface of the host application talks to the GXS service level to gather/send information related to specific services (e.g. forums, channels, etc) while the GXS core takes care of synchronising the data using connexions provided by the darknet layer.

main perfectly responsive all the time. An intermediate cache system keeps the result of the latest requests so that the user interface can retrieve them instantly.

Locally, unused data is gradually erased: we keep track of the usage of GXS identities and delete the ones that are not used after some time. GXS groups are also cleaned up of all messages that are older than a user-defined period.

Cryptographic primitives. Our implementation uses our own fork of OpenPGP-SDK[4] for PGP network node keys. PGP-authed TLS is implemented on top of `openssl` [5]. Connections between nodes are encrypted using perfect forward secrecy (Cipher string `DHE-RSA-AES256-GCM-SHA384`). All GXS group keys are 2048 bits RSA key pairs, for which we use `libcrypto`.

Plugins. In our implementation GXS allows developers to add new services. To do so, a developer needs to instance the GXS network sync and group validation components, and supply them with its own service-related data. When a new service is created, its data will spread the network of nodes who actually run the same plugins. Other nodes will simply discard the data.

[4]Originally available at `https://github.com/public/OpenPGP-SDK`

[5]See `https://www.openssl.org/`

13 Conclusion

In this paper we presented the GXS system, which allows to privately and securely distribute generic data inside Friend-to-Friend networks. It offers a unique combination of flexibility, security, robustness and extensibility.

In the near future, we will work on extending GXS group visibility, so as to favor the distribution of interesting groups to users for which no friend has subscribed the group. To do this, a turtle search [PCT04] will be used to gather information about groups that might not be visible at friend nodes. Similarly, tunnels can be used to provide virtual peers with which group synchronization can be achieved. This poses additional challenges in particular w.r.t. whether tunnels are needed for a given service, since friend might This can be partially achieved by including an information about the group that allows to conservatively accept tunnel requests if they have anything new to send. The exact synchronisation will then take place through the tunnel once it is open. Finding a proper marker however is not trivial (e.g. the time of the most recent message is not enough), since peers do not always keep messages for the same amount of time and therefore may not store the same subset of messages.

We believe that the distribution of data using GXS can be used to create a local routing table, that keeps track of which friend node relayed a piece of data signed by a particular identity. That information can then be used to reach this identity by relaying the data upward the flow. It is still unclear however how the data should be handled when the ideal route is broken because a friend node is disconnected.

Finally, our long term goal is to implement a complete social network with an user interface similar to Facebook, on top of GXS, using it as a transport layer.

14 Acknowledgments

I would like to express my gratitude to two former members of the Retroshare community: Mark Fernie and Chris Evi-Parker, who both contributed to GXS in its early design and implementation. A lot of help in testing, debugging and profiling was also offered by the users community on an every day basis.

A Code release

A full ready to use implementation of GXS is available in the Retroshare codebase: `http://github.com/retroshare/retroshare`. It runs the internal forums, channels, messaging and links services. In this repository, the GXS backend is in `libretroshare/src/gxs/`.

References

[ACBM08] Elli Androulaki, Seung Geol Choi, Steven M. Bellovin, and Tal Malkin. Reputation Systems for Anonymous Networks. In *Proceedings of the 8th International Symposium on Privacy Enhancing Technologies*, PETS '08, pages 202–218, Berlin, Heidelberg, 2008. Springer-Verlag.

[ADBS09] Jonathan Anderson, Claudia Diaz, Joseph Bonneau, and Frank Stajano. Privacy-enabling Social Networking over Untrusted Networks. In *Proceedings of the 2Nd ACM Workshop on Online Social Networks*, WOSN '09, pages 1–6, New York, NY, USA, 2009. ACM.

[AR12] Luca Maria Aiello and Giancarlo Ruffo. LotusNet: Tunable privacy for distributed online social network services. *Computer Communications*, 35(1):75–88, January 2012.

[BBS$^+$09] Randy Baden, Adam Bender, Neil Spring, Bobby Bhattacharjee, and Daniel Starin. Persona: An Online Social Network with User-defined Privacy. In *Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication*, SIGCOMM '09, pages 135–146, New York, NY, USA, 2009. ACM.

[BKB14] Oleksandr Bodriagov, Gunnar Kreitz, and Sonja Buchegger. Access Control in Decentralized Online Social Networks : Applying a Policy-Hiding Cryptographic Scheme and Evaluating Its Performance. pages 622–628, 2014.

[BSVD09] Sonja Buchegger, Doris Schiöberg, Le-Hung Vu, and Anwitaman Datta. PeerSoN: P2p Social Networking: Early Experiences and Insights. In *Proceedings of the Second ACM EuroSys Workshop on Social Network Systems*, SNS '09, pages 46–52, New York, NY, USA, 2009. ACM.

[BSW07] John Bethencourt, Amit Sahai, and Brent Waters. Ciphertext-Policy Attribute-Based Encryption. In *Proceedings of the 2007 IEEE Symposium on Security and Privacy*, SP '07, pages 321–334, Washington, DC, USA, 2007. IEEE Computer Society.

[CDG$^+$02] Miguel Castro, Peter Druschel, Ayalvadi Ganesh, Antony Rowstron, and Dan S. Wallach. Secure Routing for Structured Peer-to-peer Overlay Networks. *SIGOPS Oper. Syst. Rev.*, 36(SI):299–314, December 2002.

[CMS09] L. A. Cutillo, R. Molva, and T. Strufe. Safebook: A privacy-preserving online social network leveraging on real-life trust. *IEEE Communications Magazine*, 47(12):94–101, December 2009.

[CSWH01] Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore W. Hong. Freenet: A Distributed Anonymous Information Storage and Retrieval System. In *International Workshop on Designing Privacy Enhancing Technologies: Design Issues in Anonymity and Unobservability*, pages 46–66, New York, NY, USA, 2001. Springer-Verlag New York, Inc.

[dia] The diaspora* Project `https://diasporafoundation.org/`.

[DMS08] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. In *Proceedings of the 13th USENIX Security Symposium*, 2008.

[GMS12] Felix Günther, Mark Manulis, and Thorsten Strufe. Cryptographic Treatment of Private User Profiles. In *Proceedings of the 2011 International Conference on Financial Cryptography and Data Security*, FC'11, pages 40–54, Berlin, Heidelberg, 2012. Springer-Verlag.

[GTAL12] R. Gracia-Tinedo, M. S. Artigas, and P. García López. Analysis of data availability in F2f storage systems: When correlations matter. In *2012 IEEE 12th International Conference on Peer-to-Peer Computing (P2P)*, pages 225–236, September 2012.

[IMC10] Stratis Ioannidis, Laurent Massoulie, and Augustin Chaintreau. Distributed Caching over Heterogeneous Mobile Networks. In *Proceedings of the ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '10, pages 311–322, New York, NY, USA, 2010. ACM.

[IPKA10] Tomas Isdal, Michael Piatek, Arvind Krishnamurthy, and Thomas Anderson. Privacy-preserving P2p Data Sharing with OneSwarm. In *Proceedings of the ACM SIGCOMM 2010 Conference*, SIGCOMM '10, pages 111–122, New York, NY, USA, 2010. ACM.

[JFH] Bufford John F. and Yu Heather. *Handbook of Peer-to-Peer Networking | Xuemin Sherman Shen | Springer*.

[JNM$^+$12] S. Jahid, S. Nilizadeh, P. Mittal, N. Borisov, and A. Kapadia. DECENT: A decentralized architecture for enforcing privacy in online social networks. In *2012 IEEE International Conference on Pervasive Computing and Communications Workshops*, pages 326–332, March 2012.

[KSW08] Jonathan Katz, Amit Sahai, and Brent Waters. Predicate Encryption Supporting Disjunctions, Polynomial Equations, and Inner Products. In *Proceedings of the Theory and Applications of Cryptographic Techniques 27th Annual International Conference on Advances in Cryptology*, EUROCRYPT'08, pages 146–162, Berlin, Heidelberg, 2008. Springer-Verlag.

[MU10] Alfred Menezes and Berkant Ustaoglu. On reusing ephemeral keys in diffie-hellman key agreement protocols. *Int. J. Appl. Cryptol.*, 2(2):154–158, January 2010.

[NAHK11] Shirin Nilizadeh, Naveed Alam, Nathaniel Husted, and Apu Kapadia. Pythia: A Privacy Aware, Peer-to-peer Network for Social Search. In *Proceedings of the 10th Annual ACM Workshop on Privacy in the Electronic Society*, WPES '11, pages 43–48, New York, NY, USA, 2011. ACM.

[NJM$^+$12] Shirin Nilizadeh, Sonia Jahid, Prateek Mittal, Nikita Borisov, and Apu Kapadia. Cachet: A Decentralized Architecture for Privacy Preserving Social Networking with Caching. In *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies*, CoNEXT '12, pages 337–348, New York, NY, USA, 2012. ACM.

[NL15] Y. Nir and A. Langley. ChaCha20 and Poly1305 for IETF Protocols. RFC 7539 (Informational), May 2015.

[PCT04] B.C. Popescu, B. Crispo, and A.S. Tanenbaum. Safe and private data sharing with turtle: Friends team-up and beat the system. *Lectures notes in computer science. Security Protocols*, 3957, 2004.

[PD12] Anna-Kaisa Pietilänen and Christophe Diot. Dissemination in Opportunistic Social Networks: The Role of Temporal Communities. In *Proceedings of the Thirteenth ACM International Symposium on Mobile Ad Hoc Networking and Computing*, MobiHoc '12, pages 165–174, New York, NY, USA, 2012. ACM.

[PFS14] Thomas Paul, Antonino Famulari, and Thorsten Strufe. A Survey on Decentralized Online Social Networks. *Comput. Netw.*, 75(PA):437–452, December 2014.

[SDDM11] R. Sharma, A. Datta, M. DeH'Amico, and P. Michiardi. An empirical study of availability in friend-to-friend storage systems. In *2011 IEEE International Conference on Peer-to-Peer Computing*, pages 348–351, August 2011.

[SHS$^+$15] Russell G. Shirey, Kenneth M. Hopkinson, Kyle E. Stewart, Douglas D. Hodson, and Brett J. Borghetti. Analysis of Implementations to Secure Git for Use As an Encrypted Distributed Version Control System. In *Proceedings of the 2015 48th Hawaii International Conference on System Sciences*, HICSS '15, pages 5310–5319, Washington, DC, USA, 2015. IEEE Computer Society.

[Sol17] Cyril Soler. The retroshare Project `https://retroshare.net/`, 2017.

[TMB$^+$13] M. Taghizadeh, K. Micinski, S. Biswas, C. Ofria, and E. Torng. Distributed Cooperative Caching in Social Wireless Networks. *IEEE Transactions on Mobile Computing*, 12(6):1037–1053, June 2013.

[Tot13] Gabor Toth. *Design of a Social Messaging System Using Stateful Multicast*. PhD thesis, University of Amsterdam, Amsterdam, 2013.

[TVSP17] K. Thilakarathna, A. C. Viana, A. Seneviratne, and H. Petander. Design and Analysis of an Efficient Friend-to-Friend Content Dissemination System. *IEEE Transactions on Mobile Computing*, 16(3):702–715, March 2017.

[Wac15] Matthias Wachs. *A Secure and Resilient Communication Infrastructure for Decentralized Networking Applications*. PhD thesis, Technische Universität München, München, 2015.

[WOW08] Zooko Wilcox-O'Hearn and Brian Warner. Tahoe: The Least-authority Filesystem. In *Proceedings of the 4th ACM International Workshop on Storage Security and Survivability*, StorageSS '08, pages 21–26, New York, NY, USA, 2008. ACM.

[ZH11] Bassam Zantout and Ramzi Haraty. I2p data communication system. In *Proceedings of ICN 2011, The Tenth International Conference on Networks*, January 2011.

[ZI16] Xiang Zuo and Adriana Iamnitchi. A Survey of Socially Aware Peer-to-Peer Systems. *ACM Comput. Surv.*, 49(1):9:1–9:28, May 2016.

RESEARCH CENTRE
GRENOBLE – RHÔNE-ALPES

Inovallée
655 avenue de l'Europe Montbonnot
38334 Saint Ismier Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399