Int. J. Inf. Secur. (2017) 16:313–326
DOI 10.1007/s10207-016-0318-0

REGULAR CONTRIBUTION

Broadcast anonymous routing (BAR): scalable real-time anonymous communication

Panayiotis Kotzanikolaou[1] · George Chatzisofroniou[1] · Mike Burmester[2]

Published online: 3 February 2016

Abstract We propose BAR, a scalable anonymous Internet communication system that combines broadcasting features of dc-net with layered encryption of mix-nets. The main advantage of BAR over other broadcast systems is bandwidth configurability: by using selective broadcasting it can significantly reduce the required bandwidth for a small increase in latency, without affecting anonymity. Unlike mix-net systems, BAR provides unlinkability protection while minimizing the use of public key operations. BAR provides sender, receiver and session anonymity with forward secrecy. We analyze the efficiency of BAR for several anonymity configurations by using a prototype implementation.

Keywords Sender/receiver anonymity · Unlinkability · Dc-net · Mix-nets

1 Introduction

Anonymous communication systems allow users to communicate privately over the Internet, by hiding the relation between a message and the IP address of its sender (*sender*-anonymity) and/or its receiver (*receiver*-anonymity). They may also hide the link between different messages *e.g.,* by making it infeasible to decide whether two messages belong to the same session or not (*sender–receiver* anonymity, or *unlinkability*).

A toy example Clients communicate in a noisy crowded bar by having bar servers broadcast messages. Messages are encrypted for privacy (clients earlier exchanged secret keys). Each client trusts his/her partner and the bar server for broadcasting messages in real-time. All clients receive the encryptions. However except for the intended receiver who can identify the encryptions, the other clients cannot distinguish them from noise. An eavesdropper might try to identify sender–receiver pairs by monitoring client activities. To thwart such attempts, clients give the bar server constant size messages: either encryptions or noise (random bitstrings), at constant time periods. This provides (i) sender anonymity, since it is not possible for an eavesdropper to identify the actual sender (messages are encrypted), (ii) receiver anonymity, since it is not possible to ascertain who the intended receiver is and (iii) sender–receiver anonymity, since it is not possible to link exchanged messages.

Our goal is to design a practical and scalable system for real-time, concurrent anonymous communication for large numbers of users. We model the bar servers as overlay application layer broadcast anonymous routers, and clients are randomly assigned to BARs, subject to threshold bounds.

Contribution Our system, BAR, provides sender, receiver and sender–receiver anonymity in the presence of an *honest-but-curious* adversary. BAR combines dc-net and mix-net features. For receiver anonymity, it uses broadcast channels [5]. Compared to similar approaches (*e.g.*, [22]), BAR has lower computational costs, due to an efficient `Filter` mechanism that allows clients to selectively decrypt only those messages intended for them. More importantly, BAR has significantly lower broadcast costs. Using a selective broadcasting mechanism, it can be fine tuned to reduce

✉ Panayiotis Kotzanikolaou
pkotzani@unipi.gr

Mike Burmester
burmeste@cs.fsu.edu

[1] Department of Informatics, University of Piraeus, Piraeus, Greece

[2] Department of Computer Science, Florida State University, Tallahassee, FL, USA

bandwidth for the same level of anonymity with a small increase in end-to-end latency. For sender anonymity, a mix-net approach is used. However unlike mix-nets that usually require layers of public key encryption (*e.g.*, [4,11]), BAR mainly uses symmetric encryption. Anonymous key exchange is based on an embedded key management mechanism. For global communication between users assigned to different BARs, random session public keys are used.

Paper structure. In Sect. 2, we describe BAR and extend it for unreliable networks. In Sect. 3, we prove that it provides strong anonymity against an honest-but-curious adversary. In Sect. 4, we analyze its efficiency and we propose a selective broadcast mechanism that reduces bandwidth costs. In Sect. 5, we review related work. Section 6 concludes this paper.

2 The BAR communication protocol

The parties of BAR are: the users, the BAR servers and a system coordinator whose role is to publish system parameters and support its operation. The coordinator and the BAR servers must be available in real-time, but no other trust assumptions regarding the system are made. In Table 1 we define our notation.

2.1 The components of the protocol

Let $enc_k\{\cdot\}$ denote the encryption function with key k, $dec_k\{\cdot\}$ the decryption function with key k, $sig_{sk}(\cdot)$ the signature function with private key sk and $ver_{pk}(\cdot)$ the verification function with public key pk. Let | denote string concatenation. We define the following operators:

$\texttt{AddUser}(List_i;\ [pk_j, k_{ij}, l_{ij}]) \mapsto List_i$: adds entry $[pk_j, k_{ij}, l_{ij}]$ of u_j to $List_i$.
$\texttt{UpdateListEntry}(List_i; pk_j, k'_{ij}, l'_{ij}) \mapsto List_i$, updates entry $[pk_j, k_{ij}, l_{ij}]$ in $List_i$: $k_{ij} \leftarrow k'_{ij}, l_{ij} \leftarrow l'_{ij}$.
$\texttt{Encrypt}([pk_j, k_{ij}, l_{ij}];\ k'_{ij}, l'_{ij}, m) \mapsto \langle l_{ij},\ c_j\rangle$, with $c_j = enc_{k_{ij}}\{k'_{ij}|l'_{ij}|l_{ij}|m\}$.
$\texttt{Filter}(List_u;\ \langle l_{ij}, c_j\rangle) \mapsto \{0, 1\}$, with output 1 if $u = u_j$ is the intended receiver, and 0 otherwise.
$\texttt{Decrypt}([pk_i, k_{ij}, l_{ij}];\ \langle l_{ij}, c_j\rangle) \mapsto (k'_{ij}, l'_{ij}, l_{ij}, m)$.

2.2 Registration, login and key exchange protocols

The user address space is defined by a n-bit cryptographic hash function (*e.g.*, SHA-256). The address space is segmented into M physical partitions, and each partition is managed by one BAR server. Each server is assigned a segment of size $2^n/M$ that is published by the coordinator $\texttt{BAR}_0$. Depending on the current number of users, $\texttt{BAR}_0$ merges (or

Table 1 Entities, system parameters and notation

Entities	Description
$\mathbb{U} = \{u_1, u_2, \ldots, u_N\}$	The set of users
$\texttt{BAR}_1, \texttt{BAR}_2, \ldots, \texttt{BAR}_M$	The physical broadcast servers
$\texttt{BAR}_0$	The system coordinator: publishes system parameters and arranges the BARs in logical partitions
$Cluster_1, \ldots, Cluster_\mu$	The logical broadcast channels dynamically arranged by $\texttt{BAR}_0$. Clusters may consist of one or more adjacent BAR servers. User receive all traffic within their cluster
System parameters	**Description**
N_{min}	The *anonymity parameter*: the minimum number of users in a cluster
N_{max} $(\geq 2 \cdot N_{min})$	The *broadcast efficiency parameter*: the maximum number of users allowed by a cluster
Notation	**Description**
nym_i	A friendly pseudonym used to identify an anonymous user u_i
pk_i, sk_i	Long-term public/private key pair of u_i
k_{ij}, l_{ij}	Symmetric key and a random label of u_i, u_j, used for one exchange
$\overline{pk}_i, \overline{sk}_i$	Bridge public/private key pair of u_i, used to bridge anonymous communications between clusters; updated in each login and not related to pk_i, sk_i
$List_i$	A list maintained by u_i containing keying material $[pk_j, k_{ij}, l_{ij}]$ of users u_j for anonymous communication
$UsersList$	A public list with entries $[nym_i, pk_i, \sigma_i]$ where $\sigma_i = sig_{BAR_0}(nym_i|pk_i)$, for all the users
$ActiveList$	A public list with entries $[id_i, IP_i, \overline{pk}_i]$, where id_i and IP_i are the unique id and IP address of u_i

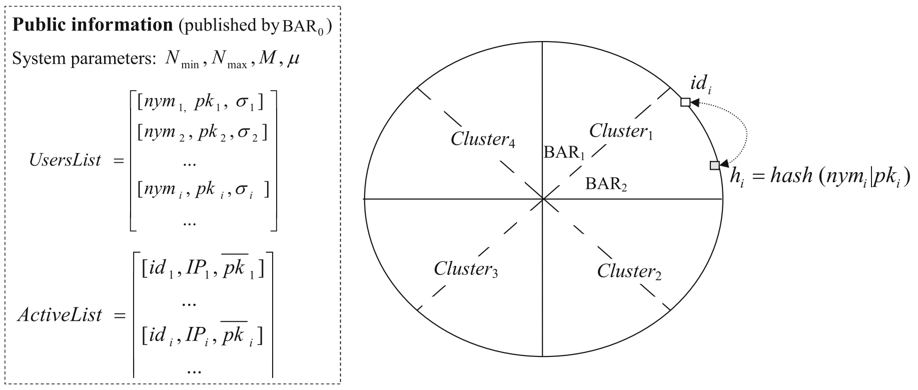

Public information (published by BAR₀)
System parameters: N_min, N_max, M, μ
UsersList = [nym₁, pk₁, σ₁] [nym₂, pk₂, σ₂] ... [nymᵢ, pkᵢ, σᵢ] ...
ActiveList = [id₁, IP₁, pk₁] ... [idᵢ, IPᵢ, pkᵢ] ...
Cluster₁
Cluster₂
Cluster₃
Cluster₄
BAR₁
BAR₂
idᵢ
hᵢ = hash (nymᵢ|pkᵢ)

Fig. 1 **User login** (with 8 BAR servers and 4 logical clusters). User u_i computes $h_i = hash(nym_i|pk_i)$ with $h_i \in \mathtt{BAR}_2$ (currently belonging to $Cluster_1$). Then u_i selects a fresh bridge public/private key pair $\overline{pk}_i, \overline{sk}_i$ and asks $\mathtt{BAR}_0$ to get assigned an $id_i \in \mathtt{BAR}_2$ for $\overline{pk}_i$. $\mathtt{BAR}_0$ chooses a random and unused id_i and adds $[id_i, IP_i, \overline{pk}_i]$ to *ActiveList*. Although h_i, id_i map to the same user u_i, this relation is only known to u_i. The entries related with u_i in *UsersList* and *ActiveList* cannot be linked

splits) physical servers to $\mu \leq M$ logical broadcast partitions (clusters), to assure that each cluster always contains from N_{min} to N_{max} users. This assures at least N_{min}-anonymity for bandwidth proportional to at most N_{max}.

User registration protocol

Users register by publishing anonymously via $\mathtt{BAR}_0$ an entry in *UserList*. User u_i sends an entry of the form $[nym_i, pk_i, \sigma_i]$ anonymously by constructing an onion routing path to the coordinator using the bridge public keys. Although this provides sender anonymity only, it is sufficient during registration.

1. User u_i
   - (a) Select a unique pseudonym nym_i and generate a public/private key pair[1] pk_i, sk_i.
   - (b) Select at random $\geq$ 3 users[2] from *ActiveList* and use their bridge keys $\overline{pk}_j$ to construct a onion routing path to the coordinator $\mathtt{BAR}_0$.
   - (c) Use this route to send nym_i, pk_i to $\mathtt{BAR}_0$.

(2) Coordinator $\mathtt{BAR}_0$
   - (a) Check if the received values are appropriate and not already in *UsersList*.
   - (b) Compute $\sigma_i = sig_{sk_{\mathtt{BAR}_0}}(nym_i|pk_i)$.
   - (c) Add $[nym_i, pk_i, \sigma_i]$ to *UsersList*.

[1] The public key and pseudonym should not be linked to the actual identity of the user or used in other services.

[2] Tor uses 3 by default; an entry, a relay and an exit node. As suggested in [2] by increasing the onion length, users may increase sender anonymity protection, with a cost in latency.

User login protocol

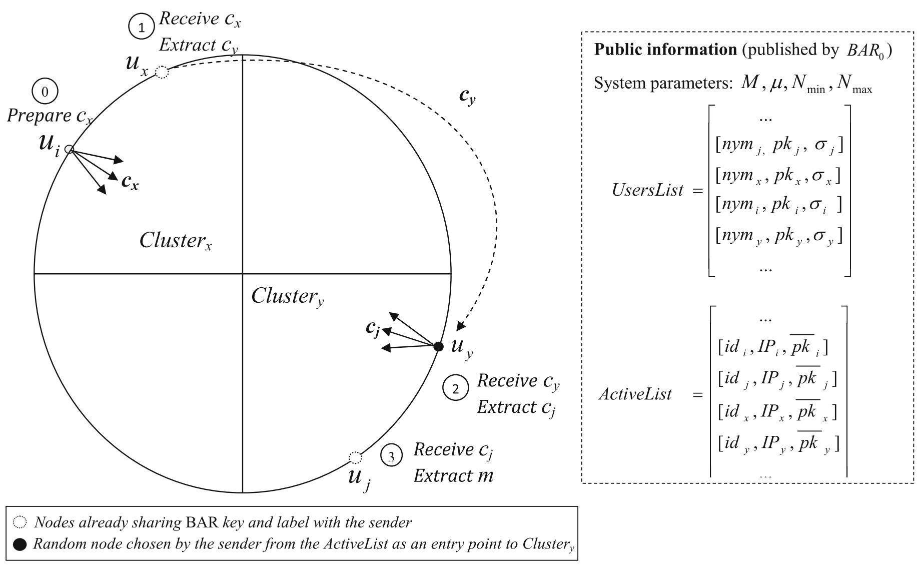
To login to the BAR system, a registered user u_i uses the following protocol (Fig. 1).

(1) User u_i
   - (a) Compute $h_i = hash(nym_i|pk_i)$. Say that $h_i \in \mathtt{BAR}_i$.
   - (b) Use the public system parameters M, μ, N_{min}, N_{max}, to find the cluster of h_i, say $h_i \in Cluster_x$.
   - (c) Select a new bridge public/private key pair $\overline{pk}_i, \overline{sk}_i$ for the current session.
   - (d) Send to $\mathtt{BAR}_0$: $IP_i, \overline{pk}_i, \mathtt{BAR}_i.Cluster_x$.

(2) Coordinator $\mathtt{BAR}_0$
   - (a) Choose a random $id_i \in \mathtt{BAR}_i$, not in use.
   - (b) Add entry $[id_i, IP_i, \overline{pk}_i]$ to the *ActiveList*.

In this protocol, u_i is assigned id_i for the current session. On logout, the coordinator removes the corresponding entry in *ActiveList*, and id_i can be assigned to another user. Note that the coordinator cannot lie about the public system parameters or assign users to wrong clusters, without being detected.

Remark 1 The coordinator continuously monitors the dynamic changes in the number of users per cluster to manage the system clustering and prevent the thresholds N_{min}, N_{max} being violated. For example, the coordinator can handoff a BAR server from one cluster to another if the threshold N_{max} is exceeded, or split a cluster into two new clusters. The range between N_{min} and N_{max} gives to the coordinator adequate flexibility. Similar handoff stabilization procedures are used in peer-to-peer systems (*e.g.*, [25]), to assure that churns in user activity do not affect the required system parameters.



0 Prepare c_y
1 Receive c_x Extract c_y
u_x
u_i
c_x
c_y
$Cluster_x$
$Cluster_y$
c_j
u_y
2 Receive c_y Extract c_j
3 Receive c_j Extract m
u_j
○ Nodes already sharing BAR key and label with the sender
● Random node chosen by the sender from the ActiveList as an entry point to $Cluster_y$
Public information (published by BAR_0)
System parameters: $M, \mu, N_{\min}, N_{\max}$
$UsersList = \begin{bmatrix} \ldots \\ [nym_j, pk_j, \sigma_j] \\ [nym_x, pk_x, \sigma_x] \\ [nym_i, pk_i, \sigma_i] \\ [nym_y, pk_y, \sigma_y] \\ \ldots \end{bmatrix}$
$ActiveList = \begin{bmatrix} \ldots \\ [id_i, IP_i, \overline{pk}_i] \\ [id_j, IP_j, \overline{pk}_j] \\ [id_x, IP_x, \overline{pk}_x] \\ [id_y, IP_y, \overline{pk}_y] \\ \ldots \end{bmatrix}$

Fig. 2 A high-level description of the BCP

Key exchange protoocol

To enable BAR communication, users must first anonymously exchange their initial pairwise keys as follows.

(1) Newly registered user u_i

  (a) For each valid entry $[nym_j, pk_j, \sigma_j]$ do:
  Choose a random label l_{ij} and key k_{ij}.
  Sign $\sigma_{ij} = sig_{sk_i}(nym_i, pk_i, pk_j, k_{ij}, l_{ij})$.
  Generate $c_{ij} = enc_{pk_j}\{nym_i, pk_i, k_{ij}, l_{ij}, \sigma_{ij}\}$.
  (b) Select ≥ 3 users from *ActiveList* at random and use the bridge keys $\overline{pk}_j$ to construct an onion routing path to $\mathtt{BAR}_0$.
  (c) Use this route to anonymously send to $\mathtt{BAR}_0$ the pairs $[nym_j, c_{ij}]$.

(2) Coordinator $\mathtt{BAR}_0$

  (a) Maintain a public bulletin board.
  (c) Publish on the bulletin board each $[nym_j, c_{ij}]$ that is received through onion route paths.

(3) Other BAR users $u_j,\ j \neq i$

  (a) At login, search the bulletin board for new entries containing nym_j.
  (b) For each new entry, use sk_j to decrypt c_{ij} and get $\{nym_i, pk_i, k_{ij}, l_{ij}, \sigma_{ij}\}$.
  (c) Verify σ_{ij}. If successful, add $[pk_i, l_{ij}, k_{ij}]$ to $List_j$.

2.3 The BAR communication protocol (BCP)

For our first communication protocol, we assume that the network is reliable, *i.e.*, send messages are received within a reasonable time frame. In Sect. 2.4, we shall extend this protocol to capture network failure.

A user u_i can anonymously send a message to a user u_j identified by the pseudonym (nym_j, pk_j), by using the following protocol (see Fig. 2 for an overview).

(1) Sender u_i

  (a) Run User Login to get assigned to $Cluster_x$.
  (b) Compute $h_j = hash(nym_j|pk_j)$ to identify the logical partition of the intended receiver u_j. Say $u_j \in Cluster_y$.
  (c) Select in *ActiveList* at random an "entry user" u_y with $id_y \in Cluster_y$ (the receiver's cluster).
  (d) Select in $List_i$ an "exit user" $u_x \neq u_i$ in $Cluster_x$ (the sender's cluster) with whom secret keys k_{ij}, l_{ij} have been exchanged, Sect. 2.2.[3]
  (e) Get entries $[pk_j, k_{ij}, l_{ij}]$, $[pk_x, k_{ix}, l_{ix}]$ from $List_i$.
  (f) Choose random keys k'_{iz} and labels l'_{iz}, $z = j, x$.
  (g) $\mathtt{Encrypt}([pk_j, k_{ij}, l_{ij}]; k'_{ij}, l'_{ij}, m)$ to get $\langle l_{ij}, c_j \rangle$.
  (h) Compute $c_y = enc_{\overline{pk}_y}\{l_{ij}|c_j\}$ using $[id_y, IP_y, \overline{pk}_y]$ in *ActiveList*.
  (i) $\mathtt{Encrypt}([pk_x, k_{ix}, l_{ix}]; k'_{ix}, l'_{ix}, (IP_y|c_y))$ to get $\langle l_{ix}, c_x \rangle$.
  (j) Broadcast $\langle l_{ix}, c_x \rangle$ to $Cluster_x$.
  (k) $\mathtt{UpdateListEntry}(List_i; [pk_z, k'_{iz}, l'_{iz}]), z = j, x$.

(2) All users $u \in Cluster_x$

  $\mathtt{Filter}(List_u; \langle l_{ix}, c_x \rangle)$: for $u \neq u_x$ the output is 0 and $\langle l_{ix}, c_x \rangle$ is dropped; for u_x the output is 1.

(3) Exit user u_x

  (a) $\mathtt{Decrypt}([pk_i, k_{ix}, l_{ix}]; \langle l_{ix}, c_x \rangle)$ to get $k'_{ix}, l'_{ix}, l_{ix}, IP_y|c_y$.

[3] There is no guarantee that u_x is online or willing to participate. If so, the sender will run again the search algorithm.

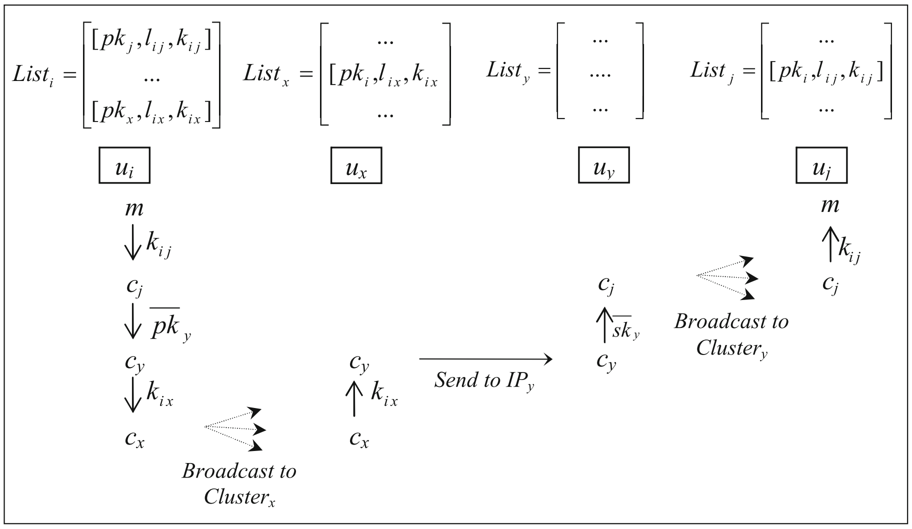


Fig. 3 Communication flows and encryptions of the BCP

(b) Send c_y to address IP_y.
(c) `UpdateListEntry`($List_x$; $[pk_i, k'_{ix}, l'_{ix}]$).

(4) Entry user u_y

(a) Decrypt c_y using private key $\overline{sk}_y$ to get $\langle l_{ij}, c_j \rangle$.
(b) Broadcast $\langle l_{ij}, c_j \rangle$ to $Cluster_y$.

(5) All users $u \in Cluster_y$

`Filter`($List_u$; $\langle l_{ij}, c_j \rangle$): for $u \neq u_j$ the output is 0 and $\langle l_{ij}, c_j \rangle$ is dropped; for u_j the output is 1.

(6) Receiver u_j

(a) `Decrypt`($[pk_i, k_{ij}, l_{ij}]$; $\langle l_{ij}, c_j \rangle$) to get k'_{ij}, l'_{ij}, l_{ij} and message m.
(b) `UpdateListEntry`($List_j$; $[pk_i, k'_{ij}, l'_{ij}]$).

Figure 3 shows the communication flows and encryptions of the BCP.

Remark 2 If the sender u_i and receiver u_j belong to the same cluster, then u_i could use inter-cluster broadcast to send the message to u_j. However in that case sender anonymity is subject to a collusion attack between the receiver and the BAR server. To guarantee the same anonymity level, the sender must run the BCP even when u_i, u_j are in the same cluster.

Remark 3 Sender u_i selects u_x, u_y in different ways: u_x is selected from $List_i$ and so u_i knows u_x's entry $[nym_x, pk_x, \sigma_x]$ in *UsersList*, but not its entry in *ActiveList* (and hence its IP address); u_y is selected in *ActiveList*, so u_i knows its entry in *ActiveList* but not in *UsersList*.

Remark 4 The exit user u_x sends the encryption c_y directly to IP_y (Steps 3a, b). A passive adversary can infer that a user in $Cluster_x$ is communicating with a user in $Cluster_y$. To address this, we can modify the protocol so that all users send at random time intervals a noise message of the same size as c_y, directly to a random *IP* from *ActiveList* belonging to a different cluster. The noise packets are filtered and eventually dropped by all users in the receiver's cluster.

2.4 E-BCP: an extension for unreliable networks

We shall extend BCP to address communication failure resulting from lost, dropped or delayed messages by allowing limited and controlled label reuse when there is failure.

In the extension, the keying material of each entry in $List_i$ has two values: the current values $K_{ij}^{cur} = (k_{ij}, l_{ij})$ used to label and encrypt (decrypt) a message that is sent (received), and $K_{ij}^{next} = (k'_{ij}, l'_{ij})$ to be used for the next message. The format of the entries in $List_i$ of user u_i is now: $[pk_j, K_{ij}^{cur}, K_{ij}^{next}]$.

When there is communication failure then, and only then, the sender will reuse the label in K_{ij}^{cur}, to avoid label desynchronization. The values of keying materials are updated in such a manner that materials shared between users at all times have at least one common (*key*, *label*) pair. Users `Filter` and `Decrypt` received messages using both K_{ij}^{cur} and K_{ij}^{next}.

Notation for the extended-BCP

Initially $K_{ij}^{cur} = K_{ij}^{next} = \emptyset$. When user u_i runs the Key Exchange Protocol in Sect. 2.2, then K_{ij}^{cur} is initialized: $K_{ij}^{cur} \leftarrow (k_{ij}, l_{ij})$, for random k_{ij}, l_{ij}. The operators in Sect. 2.1 are extended as follows. Two `UpdateListEntry` operators are defined: the first is used when receiving messages while the second when sending messages. Note that when sending messages the current values of keying material cannot be discarded unless an acknowledgement is received.

`UpdateListEntry`$^r(List_j; pk_i, k'_{ji}, l'_{ji}) \mapsto List_j$, updates entry $[pk_i, K^{cur}_{ji}, K^{next}_{ji}] \in List_j : K^{cur}_{ji} \leftarrow K^{next}_{ji}$ and $K^{next}_{ji} \leftarrow (k'_{ji}, l'_{ji})$.

`UpdateListEntry`$^s(List_i; pk_j, k'_{ij}, l'_{ij}) \mapsto List_i$, updates $[pk_j, K^{cur}_{ij}, K^{next}_{ij}] \in List_i : K^{next}_{ij} \leftarrow (k'_{ij}, l'_{ij})$.

`Encrypt`$([pk_j, K^{cur}_{ij}]; k'_{ij}, l'_{ij}, m) \mapsto \langle l_{ij},\ c_j\rangle$, with $c_j = enc_{k_{ij}}\{k'_{ij}|l'_{ij}|l_{ij}|m\}$, and $k_{ij}, l_{ij} \in K^{cur}_{ij}$.

`Filter`$(List_u; \langle l_{iu}, c_u\rangle) \mapsto \{0, 1\}$, with output 1 if $u = u_j$ is the intended receiver (l_{ij} is a label in $[pk_i, K^{cur}_{ji}, K^{next}_{ji}] \in List_j$), and 0 otherwise.

`Decrypt`$([pk_i, K^{cur}_{ji}, K^{next}_{ji}]; \langle l_{ij}, c_j\rangle) \mapsto (k'_{ji}, l'_{ji}, l_{ji}, m)$, for $k_{ji} \in K^{cur}_{ji}$ or $k'_{ji} \in K^{next}_{ji}$.

`FiltDecUpdate`$(List_u; \langle l_{ij}, c_j\rangle) \mapsto 0$ or $(List_j, dec_{k_{ij}}\{c_j\})$, filters, decrypts and updates the pk_i-entry in $List_j$:

1. If `Filter`$(List_u; \langle l_{ij}, c_j\rangle) \mapsto 0$ then drop $\langle l_{ij}, c_j\rangle$.
2. If `Filter`$(List_j; \langle l_{ij}, c_j\rangle) \mapsto 1$ then let: `Decrypt` $([pk_i, K^{cur}_{ji}, K^{next}_{ji}]; \langle l_{ij}, c_j\rangle) \mapsto (k'_{ji}, l'_{ji}, l_{ji}, m)$. If $l_{ij} \neq l_{ji}$ then drop $\langle l_{ij}, c_j\rangle$ and abort. Else:
   (a) If $l_{ij} \in K^{cur}_{ji}$ then $K^{cur}_{ji} \leftarrow (k'_{ji}, l'_{ji})$.
   (b) If $l_{ij} \in K^{next}_{ji}$ then `UpdateListEntry`^{r} $(List_j; pk_i, k'_{ji}, l'_{ji}) \mapsto List_j$.

Observe that if the label $l_{ij} \in K^{cur}_{ji}$ then it was used earlier by u_i, but the message was delayed/lost. In this case, only K^{cur}_{ji} is updated by the receiver u_j. If l_{ij} is in K^{next}_{ji} then it is fresh. In this case, we have a full update of the keying materials.

The extended BCP (E-BCP)

We use the same enumeration as in the BCP.

0. All users
   `FiltDecUpdate` any received message.
1. Sender u_i
   (a) –(j) of BCP with the extended operators.
   (k) `UpdateListEntry`$^s(List_i; [pk_z, k'_{iz}, l'_{iz}])$, $z = j, x$.
2. All users $u \in Cluster_x$
   `FiltDecUpdate`$(List_u; \langle l_{ix}, c_x\rangle)$ and drop $\langle l_{ix}, c_x\rangle$.
3. Exit user u_x
   Send c_y to IP_y (`FiltDecUpdate` will output c_y).
4. Entry user u_y As in BCP.
5. All users $u \in Cluster_y$
   `FiltDecUpdate`$(List_u; \langle l_{ij}, c_j\rangle)$ and drop $\langle l_{ij}, c_j\rangle$.
6. Receiver u_j
   `FiltDecUpdate`$(List_j; \langle l_{ij}, c_j\rangle)$ with output m and the correctly updated $List_j$.

Remark 5 E-BCP can be fine tuned to allow up to f consecutive failures *without* label reusability as follows: for the t-th failure, $0 \le t \le f$, the sender uses K^{cur}_{ij} to get the pair: $(k^{(t)}_{ij} = k_{ij}, l^{(t)}_{ij} = hash(k_{ij}|\ l_{ij} + t))$. The labels $l^{(t)}_{ij}$ are pseudo-random if k_{ij} is secret. Users `Filter` using all $f + 2$ labels $l^{(t)}_{ij}, l'_{ij}$ and `Decrypt` using the keys k_{ij}, k'_{ij}, in K^{cur}_{ij}, K^{next}_{ij} respectively.

3 Security analysis

We consider a typical TCP overlay network. Broadcast channels are implemented via dedicated servers, which maintain persistent connections with the clients and whose role is to broadcast all the received traffic to all the connected users. The IP address of each active user is in *ActiveList* and the anonymous public key in *UsersList*. However the link between entries $[id_i, IP_i, \overline{pk_i}]$ and $[nym_i, pk_i, \sigma_i]$ of the same user is only known to the user.

Threat model Honest users are those that adhere to the protocol, while compromised users are those that collude with the adversary and leak their keying material. We assume a global *honest-but-curious* adversary: a passive eavesdropper that monitors all communication channels and has access to private keys of compromised users, say $\mathcal{K}$. The adversary can also perform active attacks on the traffic, such as injecting watermarked ingress flows. The goal of BAR is to provide anonymity among the set of the honest users. To capture realistic threat scenarios, we consider three types of adversary: a sender adversary $\mathcal{A}^s$, a receiver adversary $\mathcal{A}^r$ and a sender–receiver adversary $\mathcal{A}^{sr}$. The adversary is modeled by a probabilistic polynomial time Turing machine (PPT).

Trust assumptions The BAR servers (including the coordinator) provide a real-time reliable service. In particular, the integrity and real-time availability of the public lists *UsersList*, *ActiveList* is assumed. Finally, we assume that each BAR server may contain a large fraction of compromised users, but it also contains a fraction of honest users.

3.1 Sender anonymity

$\mathcal{A}^s$ monitors the communication of the BAR users and servers and has access to the set $\mathcal{K}$ of keys of compromised users. The goal of $\mathcal{A}^s$ is to find the address IP_i of the sender of a target message $\langle l_{ij}, c_j\rangle$, by identifying the entry $[id_i, IP_i, \overline{pk_i}]$ of u_i in *ActiveList*.

Let Π be the BAR communication protocol. We formalize sender anonymity by an experiment $Priv_{\mathcal{A}^s,\Pi}(n)$ in which $\mathcal{A}^s$ has access to an oracle $\mathcal{O}^s$ that on input the security parameter n (*e.g.*, the length of the hash function), simulates executions of Π. $\mathcal{A}^s$ obtains from $\mathcal{O}^s$ the public system parameters, the compromised keys $\mathcal{K}$, and a history of simulated executions of Π that includes *UsersList*, *ActiveList*, and the transmitted messages (but not the links) for any session. For the test, $\mathcal{O}^s$ simulates a new session $\pi_{u_i u_j}$ for a particular sender–receiver pair u_i, u_j for which u_i is not compromised. $\mathcal{A}^s$ is given *UsersList*, *ActiveList*, the pseudonym of the sender $[nym_i, pk_i, \sigma_i]$, the communication of session $\pi_{u_i u_j}$, and must find the address IP_i of the sender. If $\mathcal{A}^s$ succeeds then $Priv_{\mathcal{A}^s,\Pi}(n)$ outputs 1, otherwise it outputs 0.

Definition 1 Π provides *sender anonymity* if: $\forall$ PPT adversary $\mathcal{A}^s$, $\exists$ a negligible function *negl* such that:

$$\text{advantage}(\mathcal{A}^s) = |Pr[Priv_{\mathcal{A}^s,\Pi}(n) = 1] - 1/N| = negl(n),$$

where N is the number of non-compromised users in the cluster of the sender. (The probability is taken over the coin tosses of $\mathcal{A}^s$.) We say that Π is N*-sender anonymous*.

Theorem 1 *Suppose the communication channels and* BAR *servers are reliable. Then BCP provides sender anonymity if at least one of the bridge users u_x or u_y is not compromised.*

Proof By the definition of sender anonymity, the sender u_i is not compromised. We shall assume that the receiver u_j is compromised, so its keys are in $\mathcal{K}$: note that $\mathcal{A}^s$ can do no better when u_j is not compromised.

$\mathcal{A}^s$ cannot use traffic analysis to identify u_i because all users send constant size messages at constant rate. However $\mathcal{A}^s$ can identify the active users in $Cluster_x$ from the public system parameters (the address space partitioning) and *ActiveList*, as well as the entries in *UsersList* that belong to $Cluster_x$ by computing $h_x = hash(nym_x|pk_x)$. $\mathcal{A}^s$ also has access to all encryptions sent by users in $Cluster_x$ from the communication history. To identify the sender u_i of a target message m send to u_j during a protocol run $\pi_{u_i u_j}$, $\mathcal{A}^s$ must link the chain $m \to c_j \to c_y \to c_x$ (see Fig. 3).

Case 1: Only u_x compromised. Then $\mathcal{A}^s$ does not know the secret keys of u_y. $\mathcal{A}^s$ starts from m and uses l_{ij}, k_{ij} to link m to c_j. From $List_j$, $\mathcal{A}^s$ gets (nym_i, pk_i). Since u_x is compromised, $k_{ix} \in \mathcal{K}$, and thus $\mathcal{A}^s$ can link c_x (broadcast by u_i) to c_y (obtained from u_x). However $\mathcal{A}^s$ cannot link c_y to c_j since he does not know the key $\overline{sk}_y$ of u_y. Therefore $\mathcal{A}^s$ will fail to link the actual sender to m. Any user in $Cluster_x$ can be the sender with equal probability.

Case 2: Only u_y is compromised. Then $\mathcal{A}^s$ does not know the secret keys of u_x, in particular k_{ix} in $List_x$. $\mathcal{A}^s$ starts from m and uses l_{ij}, k_{ij} to link it to c_j. $\mathcal{A}^s$ uses $List_j$ to find the anonymous public key of the sender (nym_i, pk_i), and the history to find that c_j was broadcast to $Cluster_y$ by u_y with IP_y (the BAR server provides this information). Now $\mathcal{A}^s$ uses $\overline{sk}_j$ (obtained from u_y) to link c_j to c_y. $\mathcal{A}^s$ finds from the history that u_y received c_y from the user with address IP_x. Since u_x is not compromised, $\mathcal{A}^s$ cannot link the message c_x broadcasted by the sender, to c_y. Again $\mathcal{A}^s$ will fail.

Case 3: u_x, u_y are compromised. Then $\mathcal{A}^s$ knows the secret keys of u_j, u_x and u_y. In this case $\mathcal{A}^s$ will succeed in linking the chain $m \to c_j \to c_y \to c_x$. First $\mathcal{A}^s$ uses l_{ij}, k_{ij} to link m to c_j; then $(nym_i, pk_i) \in List_j$ and the history of exchanged messages, to find that c_j was broadcast by u_y to $Cluster_y$ with IP_y. Now $\mathcal{A}^s$ uses the private key $\overline{sk}_j$ of u_j to link c_j to c_y, and from the history finds that u_y received c_y from the user with address IP_x. Since u_x is compromised, $\mathcal{A}^s$ knows k_{ix} and thus finds that c_x was broadcast by u_i, and thus linked to c_y. Now $\mathcal{A}^s$ uses the history to find u_i's address IP_i.

3.2 Receiver anonymity

Adversary $\mathcal{A}^r$ is similar to $\mathcal{A}^s$ except that the goal is find the IP address of the receiver. Let Π be the BAR communication protocol. We formalize receiver anonymity by an experiment $Priv_{\mathcal{A}^r,\Pi}(n)$ in which $\mathcal{A}^r$ has access to an oracle $\mathcal{O}^r$ that on input the security parameter n simulates executions of Π. $\mathcal{A}^r$ obtains from $\mathcal{O}^r$ the system parameters, the compromised keys $\mathcal{K}$, and a history of simulated executions of Π that includes *UsersList*, *ActiveList* and the transmitted messages for any session. For the test, $\mathcal{O}^r$ simulates a new session $\pi_{u_i u_j}$ for a particular sender–receiver pair u_i, u_j for which u_j is not compromised. $\mathcal{A}^r$ is given *UsersList*, *ActiveList*, the pseudonym of the receiver $[nym_j, pk_j, \sigma_j]$, the communication of session $\pi_{u_i u_j}$ and must find the address IP_j of the receiver. If $\mathcal{A}^r$ succeeds then $Priv_{\mathcal{A}^s,\Pi}(n)$ outputs 1, otherwise it outputs 0.

Definition 2 Π provides *receiver anonymity* if: $\forall$ PPT adversary $\mathcal{A}^r$, $\exists$ a negligible function *negl* such that:

$$\text{advantage}(\mathcal{A}^r) = |Pr[Priv_{\mathcal{A}^r,\Pi}(n) = 1] - 1/N| = negl(n),$$

where N is the number of non-compromised users in the cluster of the receiver. We say that Π is N*-receiver anonymous*.

Theorem 2 *Suppose the communication channels and BAR servers are reliable. Then BCP provides receiver anonymity.*

Proof By the definition of receiver anonymity, the receiver u_j is not compromised. We shall assume that the sender u_i is compromised. As in the previous case $\mathcal{A}^r$ cannot use traffic analysis to identify a receiver, but can easily identify the entries in *ActiveList* and *UsersList* that belong to $Cluster_y$. $\mathcal{A}^r$ also has access to the history of received encryptions by users in $Cluster_y$. To identify the receiver of a target message m send by u_i, $\mathcal{A}^r$ must be able to re-construct the chain $c_x \to c_y \to c_j \to m$ of instance $\pi_{u_i u_j}$ and link it to the receiver. $\mathcal{A}^r$ can re-construct this chain since it has access to the secret keys of u_i, u_x, u_y. However this is not sufficient, because of the broadcast nature of the receiver's channel (Fig. 3): since all active users of $Cluster_y$ receive c_j, $\mathcal{A}^r$ cannot distinguish the actual receiver from any other receiver in $Cluster_y$ (*i.e.*, identify the entry $[id_j, IP_j, \overline{pk}_j] \in ActiveList$ of the receiver). Any active user in $Cluster_y$ is equally likely to be the receiver.

3.3 Sender–receiver anonymity

Adversary $\mathcal{A}^{sr}$ is similar to $\mathcal{A}^s$ except that the goal is to link encryptions exchanged by a sender–receiver pair u_i, u_j. Let Π be the BAR communication protocol. We formalize sender–receiver anonymity of Π by an experiment $Priv_{\mathcal{A}^{sr},\Pi}(n)$ in which $\mathcal{A}^{sr}$ has access to oracle $\mathcal{O}^{sr}$ that on input the security parameter n simulates executions of Π. $\mathcal{A}^{sr}$ obtains from $\mathcal{O}^{sr}$ the public system parameters, the compromised keys $\mathcal{K}$, and a history of simulated executions of Π that includes *UsersList*, *ActiveList* and the transmitted encryptions for any session. For the test, $\mathcal{O}^{sr}$ simulates a new session $\pi_{u_i u_j}$ for a non-compromised sender–receiver pair u_i, u_j. $\mathcal{A}^{sr}$ is given *UsersList* and *ActiveList*, including the entries for the actual pair, the communication of $\pi_{u_i u_j}$, and must decide if earlier encrypted messages $\{\langle l_{ij}, c_j\rangle\}$ are linked to entries u_i, u_j in *UsersList*. Note that $\mathcal{A}^{sr}$ is not required to find the actual users (their IPs). If $\mathcal{A}^{sr}$ succeeds then $Priv_{\mathcal{A}^{sr},\Pi}(n)$ outputs 1, otherwise it outputs 0.

Definition 3 Π provides *sender–receiver anonymity* if: $\forall$ PPT adversary $\mathcal{A}^{sr}$, $\exists$ a negligible function *negl* such that:

$$\mathsf{advantage}(\mathcal{A}^{sr}) = |Pr[Priv_{\mathcal{A}^{sr},\Pi}(n) = 1] - 1/N| = negl(n),$$

where N is the number of non-compromised active users. We say that Π is N-*sender–receiver anonymous*.

Theorem 3 *Suppose the communication channels and BAR servers are reliable. Then BCP provides sender–receiver anonymity.*

Proof By definition u_i, u_j are not compromised. Since all users broadcast encrypted messages of constant size at constant rate, it is not possible for $\mathcal{A}^{sr}$ to distinguish noise messages from actual messages. Similarly $\mathcal{A}^{sr}$ cannot identify actual bridge messages, since all users send simulated bridged messages at random time intervals to random receivers (Remark 4, Sect. 2.3). Thus even if the actual bridge users u_x, u_y of the target message are compromised, at best $\mathcal{A}^{sr}$ will learn that either a user from $Cluster_x$ is communicating with a user in $Cluster_y$ or that a simulated bridge message was sent, with equal probability.

3.4 Session anonymity

Session anonymity addresses privacy of a completed session, *i.e.*, during which a message is implicitly acknowledged and keys have been properly updated. In the extended BCP, the keying materials of the sender u_i, the bridge user u_x and the receiver u_y are updated in Step 0 by using $\texttt{UpdateListEntry}^r$. If a session is completed then u_i, u_x, u_y will receive confirmations and the keying materials are updated with fresh values, so messages from earlier sessions cannot be linked to them. In this case E-BCP reduces to BCP. Otherwise the same labels are used. In this case messages clearly can be linked.

Let Π' be the extended BCP. To formally capture session anonymity [3], we shall extend the definitions in Sects. 3.1, 3.2 and 3.3.

Definition 4 Π' provides *session sender (receiver or sender–receiver) anonymity* if the advantage of $\mathcal{A}^s$ in Definition 1 ($\mathcal{A}^r$ in Definition 2, or $\mathcal{A}^{sr}$ in Definition 3) in experiment $Priv_{\mathcal{A}^s,\Pi}(n)$ ($Priv_{\mathcal{A}^r,\Pi}(n)$, or $Priv_{\mathcal{A}^{sr},\Pi}(n)$) is negligible for any test with communication history separated by a completed session.

Theorem 4 *E-BCP provides session sender, receiver and sender–receiver anonymity.*

Proof If there is communication failure, then a sender in E-BCP will reuse the previous label. Clearly messages with the same label are linkable. However once a session is completed (the sender receives confirmation) then the shared keying materials are updated with random values. Thus the adversary cannot find additional information involving the secret keys of the sender (or receiver) prior to the completed session. After a completed session E-BCP is essentially the same as BCP and the proof that we get session sender, receiver and sender–receiver anonymity reduces to the proof of Theorems 1, 2 and 3.

3.5 Forward secrecy

Forward secrecy [15] addresses the privacy of messages that are exchanged prior to key compromise.

Definition 5 Let Π be an anonymous communication protocol and x_i, x_j a sender–receiver pair. Suppose that the keying material of user u_z gets compromised at time t_z (we allow for $t_z = \infty$). The definitions in Sects. 3.1, 3.2 and 3.3 are extended to capture forward secrecy. Π provides *forward sender (receiver, or sender–receiver) secrecy* if the advantage of $\mathcal{A}^s$ in Definition 1 ($\mathcal{A}^r$ in Definition 2, or $\mathcal{A}^{sr}$ in Definition 3) in the experiment $Priv_{\mathcal{A}^s,\Pi}(n)$ ($Priv_{\mathcal{A}^r,\Pi}(n)$, or $Priv_{\mathcal{A}^{sr},\Pi}(n)$) is negligible for any test with communication history prior to t_i (t_j, or $\min\{t_i, t_j\}$).

Π provides *session forward sender (receiver, or sender–receiver) secrecy* if the advantage of $\mathcal{A}^s$ in Definition 1 ($\mathcal{A}^r$ in Definition 2, or $\mathcal{A}^{sr}$ in Definition 3) in experiment $Priv_{\mathcal{A}^s,\Pi}(n)$ ($Priv_{\mathcal{A}^r,\Pi}(n)$, or $Priv_{\mathcal{A}^{sr},\Pi}(n)$) is negligible for any test with communication history separated by a completed session prior to t_i (t_j, or $\min\{t_i, t_j\}$).

Theorem 5 *BCP provides forward sender, receiver and sender–receiver secrecy. E-BCP provides session forward sender, receiver and sender–receiver secrecy.*

Proof For BCP this reduces to the proofs of Theorems 1, 2 and 3, by observing that a fresh (*label, key*) pair is used for each message exchange. A similar argument holds for E-BCP, since a label is reused *only* when a session is not completed.

3.6 Other security characteristics

Besides the formal analysis of anonymity properties, we examine other security characteristics of BAR.

3.6.1 Resistance to positioning attacks

Since the clustering parameters are public, malicious users may join a different cluster than the one they actually belong to, in order to position themselves to a target cluster of either the sender or the receiver.

Positioning attacks against the sender. An attacker positioning himself in the cluster of the sender, will *never* be selected as an "exit user" in the protocol by any user in the sender's cluster. Recall from Sect. 2.3 that a sender u_i chooses an exit user u_x from its $List_i$ (which in turn has been constructed based on the *UsersList*) and not from *ActiveList*. Thus u_i will never select a malicious user who has falsely positioned himself in the sender cluster in *ActiveList*, but whose (*nym*, *pk*) pair does not actually correspond to the target cluster.

Positioning attacks against the receiver. An attacker positioning himself in the receiver's cluster may be chosen by the sender as the "entry user" u_y in the protocol, since u_y is randomly selected from *ActiveList*. However recall from Sect. 3.2 (Theorem 2) that a malicious entry user cannot distinguish who is the actual receiver, due to the broadcast nature of the receiver's channel. Even in the extreme case where *all* the users in the receiver's cluster except u_j are malicious they still cannot deduce that u_j is an actual receiver, since u_x may have sent a simulated noise bridging message (see Remark 4).

3.6.2 Resistance to users with multiple identities

Malicious users may create multiple public keys and pseudonyms that belong to a cluster. Then the attacker will spoof multiple IP addresses in order to join multiple times in the target cluster and thus reduce its anonymity level. Although such attacks cannot be completely prevented, the protocol is "inherently" resistant to multiple identity attacks, due to its broadcast nature. Note that such an attacker will have to receive the broadcast traffic of its cluster as many times as its identities. This limits the capabilities of a user to present multiple accounts, since the user will practically cause a Denial-of-Service attack to herself due to the large amount of the received traffic. Obviously, filtering the traffic *after* this is received is of no use for the attacker.

3.6.3 Resistance to active traffic analysis attacks

Due to its broadcast nature, BAR is not vulnerable to active traffic analysis attacks. For example, it is not possible for an attacker to inject a watermarked ingress flow and attempt to observe it from egress flows in order to pair them.

3.6.4 Loosening trust assumptions

The security analysis of BAR relies on the trust assumptions of a reliable delivery service and trusted servers. The assumption of reliable delivery (*i.e.*, that servers will always broadcast all messages with the correct order) can be substantially reduced, if the extended-BCP protocol described in Sect. 2.4 is used. Recall that the extended-BCP allows for controlled message failures. The assumption of trusted servers can be loosened by combining local verification and global reputation mechanisms. Since each BAR server is expected to broadcast all messages to all users, it is easy for every user to locally verify whether a server correctly broadcasts its packets to others or not, by inspecting the received broadcast traffic. Users may also test the server's behavior by encrypting messages that are expected to bounce back to themselves. Then a reputation mechanism can be constructed in which users can rate the BAR servers based on their local verification. BAR servers with low reputation could be excluded from the system for a certain period or permanently.

3.6.5 Distributing the role of the BAR servers

A possible extension would be to distribute the role of the BAR servers among the users, who could collaboratively broadcast packets within clusters. Such a design would raise security issues related with corrupted or malicious users not properly forwarding packets and would require further security analysis. One way to deal with untrusted behavior is to use reputation-based mechanisms.

4 Implementation and efficiency analysis

To analyze the bandwidth and end-to-end delivery time for different numbers of users, we implemented a prototype[4] BAR server. Each client logs in and maintains a persistent TCP connection with the BAR server in order to send and receive BAR traffic. Each client sends at a constant rate its encrypted messages (real or noise) to the BAR server. The server broadcasts all the received traffic to all its clients, through the established connections. We used TCP at the transport layer in order to simulate the use of the protocol for TCP-based application layer protocols such as HTTP. Furthermore, the use of TCP simplifies issues related with packet loss.

The software is implemented in Python using the Twisted networking library. We use SQLite to store the users' lists and database indexes to improve the required search operation for the labels. All transmitted messages are encoded in the form of netstrings to avoid illegal payloads (*e.g.*, messages exceeding the maximum allowed length). The tests were performed on a Gigabit Ethernet LAN with three 2.13 GHz machines to simulate all the BAR servers and the users. We used 32-bit random labels and 128-bit AES encryption keys.

4.1 Inter-cluster analysis

Initially, we analyzed the broadcast bandwidth and delivery time within a single cluster implemented with one BAR server.

For the end-to-end delivery time, we measured all the delays imposed by the protocol from the preparation until the receipt of the message, including the time required for the following: the encryption of a message with the keying material of the receiver; sending of the message from the sender to the BAR server; the receipt and broadcasting of the message from the BAR server to all the users; the receipt and filtering of the broadcast traffic by the intended receiver; and the decryption of a successfully filtered message.

We ran 4 scenarios for N=100–400 users, assuming $N/2$ communicating pairs (every user is either a sender or a receiver). For each scenario, we ran 50 tests and computed the mean time. In all tests, all users continuously send messages (real or noise) at a constant period of 290 ms, while the message length was 0.6 KB (for actual message size 0.5 KB plus 0.1 KB used for the labels, keys and reserved bits). This leads to a send rate of about 16.5 Kbps (2.6 KBps).

[4] https://github.com/sophron/BAR/archive/master.zip.

Table 2 Inter-cluster delivery time for different user numbers

Test #	BAR users (N)	Concur. pairs	Broadcast bandwidth (Mbps/MBps)	Delivery time (ms)
1	100	50	1.62/0.2	550
2	200	100	3.23/0.4	720
3	300	150	4.85/0.6	1320
4	400	200	6.46/0.8	1400

Table 2 shows the required bandwidth and the mean delivery time within the cluster, for all the scenarios. The delivery time is between 550 and 1400 ms. The time almost exclusively depends on the broadcast bandwidth, while the computation costs slightly contribute to it. This is due to the filtering mechanism that allows clients to filter out several Mbps of broadcast traffic with minimal computational costs. For inter-cluster traffic, each user performs only one decryption per broadcast period (noise packets are filtered and dropped). On the other hand, the bandwidth cost is high, proportional to the number of users.

4.2 Bandwidth optimizations: selective broadcasting

To deal with high bandwidth costs, we propose optimizations that reduce broadcast traffic under a much smaller increase in the end-to-end delivery time, for the same anonymity parameter.

To achieve this, we exploit the controlled label reusability of E-BCP. Assume that each BAR server uses a broadcast parameter $0 < q < 1$. At each broadcast period, the server randomly selects and broadcasts $q\,N$ out of the N received messages; the remaining $(1-q)\,N$ messages are dropped.

The senders of the non-selected messages will treat this as network failure and resend messages using the previous label. Thus every message will require, on average, q^{-1} broadcasts.

Let $0 < p < 1$ be the fraction of users sending real traffic so that $(1-p)N$ users send noise messages. We consider two cases for the non-selected messages: real or noise messages. The probability of a real message being dropped by the server is:

$$Pr[\text{real message} \in \{\text{dropped message}\}] \; = \; p(1-q)$$

Each real message requires on average an additional delay of $p\,q^{-1}(1-q)$ broadcasts. Let $t_c(N)$ denote the expected inter-cluster delivery time for a cluster of size N and t_b the time required for the operations of the bridging user in E-BCP (public key decryption and packet forwarding). Then the expected end-to-end delivery time for an actual message is:

$$t_{end} = 2\,[1 + p\,q^{-1}(1-q)]\,t_c(qN) + t_b. \tag{1}$$

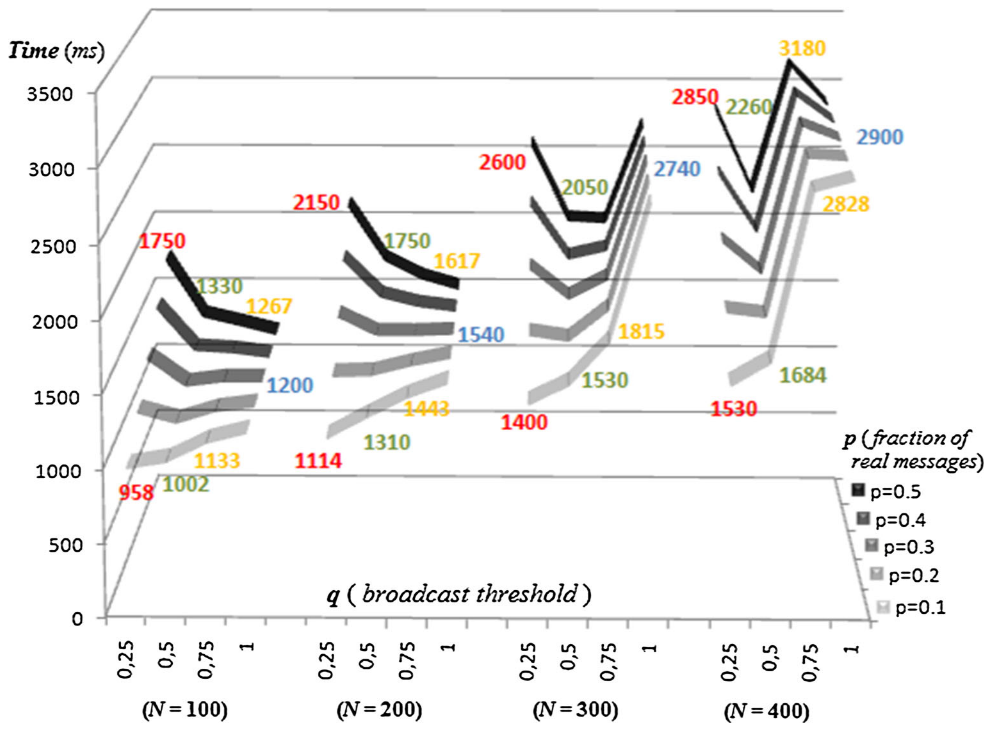


Fig. 4 End-to-end delivery time for all test scenarios

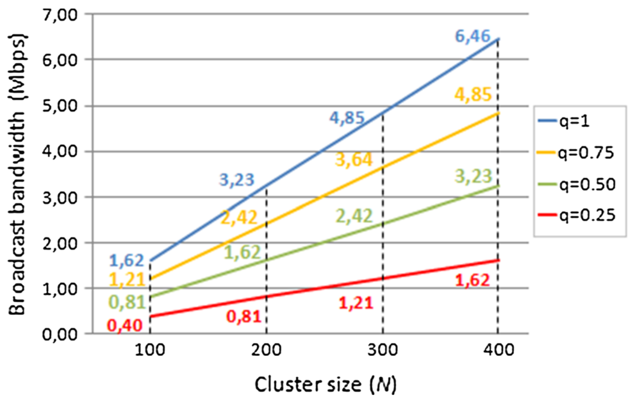


Fig. 5 Required broadcast bandwidth in relation to q

Note that the advantage gained is due to: (i) the $(1-p)(1-q)$ noise messages that are dropped, which do not impose any delay, and (ii) the reduction of the inter-cluster delivery time from $t_c(N)$ to $t_c(qN)$. More importantly, the bandwidth is significantly reduced. For send rate s Kbps, users will dedicate only qsN (instead of sN) Kbps of their bandwidth, to gain N-anonymity.

4.3 Efficiency analysis of the E-BCP

Using Eq. 1 and the inter-cluster delivery time computed in Sect. 4.1, we analyze the efficiency of E-BCP for various scenarios. Figure 4 presents the end-to-end delivery time for $N = \{100, 200, 300, 400\}$ and for varying values of $p = \{0.1, 0.2, 0.3, 0.4, 0.5\}$ and $q = \{0.25, 0.5, 0.75, 1\}$. In all tests, we used $t_b = 100$ ms.

We also measured the required bandwidth in relation to q (Fig. 5). Selective broadcasting provides effective trade-offs between the required bandwidth and the end-to-end latency. For full broadcast ($q = 1$) the delivery time depends only on the size N of the cluster, regardless of p, while the required bandwidth is proportional to N. As shown in Fig. 4, for $q = 0.25$ and various values of p, the end-to-end time is close (in some cases even lower) to the case of full broadcast, and at the same time the required bandwidth is substantially reduced. This is due to the fact that although for smaller q each message will require more broadcasts on average, at the same time the actual cluster size, and consequently cost per broadcast, is reduced to $t_c(qN)$ from $t_c(N)$. For example, for 400-anonymity, the required bandwidth per user can be reduced from about 6.5 Mbps for no optimization ($q = 1$) to only 1.6 Mbps for $q = 0.25$, with a delivery time between 1530 and 2850 ms, for p between 0.1 and 0.5. If less users are actually communicating, then the delivery time is reduced. Notice that for the selected parameters, latency is even lower than the case of full broadcasting (2900 ms). It is easy to fine-tune system parameters for the same cluster (and thus anonymity) size, by using the bandwidth versus time ratio.

Table 3 Comparison with related work

System	Anonymity properties			Real-time	Scalable	Distributed	Implemented
	Sender	Receiver	Sender–receiver				
Anonymizer [27]	Y[1]	N	N	Y	N	N	Y
Mixmaster, Mixminion [10]	Y	Y[2]	Y[4]	N	Y	Y	Y
BitMessage [28]	Y	Y	N	N	Y	Y	Y
Buses [1] and variations [16,17]	Y	Y	Y	N	N	Y	N
Cocaine anonymous broadcast [24]	Y	Y	Y	N	N	N	N
Crowds [19]	Y	N	N	Y	Y	N	Y
DC-net [5]	Y	Y	Y	N	N	Y	N
Dissent [7]	Y	Y	Y	N	N	Y	Y
Drunk Motorcyclist [29]	Y	Y	Y	N	Y	Y	N
Hordes [23]	Y	N	N	Y	Y	Y	Y
HORNET [6]	Y	N	N	Y	Y	Y	Y
P^5 [22]	Y	Y	Y	Y	Y	Y	Y
Tarzan [13]	Y	Y[3]	N	Y	Y	Y	Y
Verdict [8]	Y	Y	Y	Y	N	Y	Y
Xor-trees [12]	Y	Y	Y	N	N	Y	Y
BAR	Y	Y	Y	Y	Y	Y	Y

Comments
(1) Assuming a trusted proxy
(2) Assuming a nym server
(3) Using the hidden services protocol
(4) Excluding long-term attacks

5 Overview and comparison with related work

Most known and widely accepted architectures for anonymous online communication can be divided in two main categories [20] (Table 3): those based on mix nets [4] and those based on dc-net [5].

Although dc-nets provide full anonymity (*i.e.*, sender, receiver and unlinkability) against powerful adversaries, only one user can send a message at a time, making these systems unscalable, while it is possible for the participants to identify when a message was actually sent. Xor-trees [12] are a computationally efficient system based on dc-nets. However again only one pair of users can communicate at a time.

The Anonymizer [27] is a simple solution that provides sender anonymity, using a trusted proxy. The Anonymizer replaces information in the packet headers to spoof the IP address of the sender. Obviously, the proxy server must be trusted not to track all user activities, and hence, it is a single point of failure.

Mixminion [10] is based on mix nets, *i.e.*, on the reshuffling encrypted messages for sender anonymity. Receiver anonymity is based on a nym server. Although Mixminion makes use of noise traffic between mixes, it does not use end-to-end noise and thus is susceptible to long-term correlation attacks of a global adversary. Tor [11], Hordes [23] and Torsk [18] and HORNET [6] are systems based on onion routing [14]. They employ application layer overlay routing and public key cryptography, to provide sender anonymity. However they do not provide unlinkability and are not secure against a global passive observer who can trace packets and mount traffic analysis attacks.

Crowds [19] is a system based on blending user traffic into a crowd of other users, making it difficult for an eavesdropper to tell whether the user is the actual sender, or a router of a message. The system provides only sender anonymity, since the receiver must be known to all intermediate nodes. In Tarzan [13] and MorphMix [21], all participants generate their own traffic, or relay traffic for others using layered encryption. However both systems encounter some practical problems [9,26].

Systems based on broadcasting are generally known to be secure against active attacks on traffic, for instance watermarking attacks into traffic flows. Bitmessage [28] is a peer-to-peer system that allows users to send and receive messages by subscribing to broadcasts and provides sender and receiver anonymity. As with BAR, all users receive all broadcast messages. However in Bitmessage each user decrypts every received message, without leveraging the `Filter` mechanism of BAR. Moreover it is only intended for anonymous email exchange and may not be suitable for real-time, low end-to-end latency communications.

Dissent [7] offers sender, receiver and sender–receiver anonymity, using dc-net and verifiable shuffle algorithms to transmit variable-length messages. Verdict [8] is also based on the verifiable dc-net primitive. It uses zero-knowledge proofs of knowledge to proactively exclude misbehaving users *before* jamming the communication. The proactive exclusion of insider disruption attacks relieves the system from the need to trace a disruptor after the attack; for example in Dissent, tracing a disruptor in a group of 1000 users will require more than 60 min. In contrast to traditional dc-net, Verdict relies on public key cryptography for message encryption, which increases the computation cost.

Verdict is suitable for low-latency communications for small groups of users. However it does not scale well. According to [8], the latency for 100 users in microblogging applications is 1 s and increases to 10 s for 1000 users. BAR has higher latency but is far more scalable, since the latency does not depend on the number of active users, but only on the anonymity parameters N_{min}, N_{max}. If we consider clusters of size 100 users, each user would gain 100 anonymity for a mean latency of 1.3 s. In the above example, if 10 clusters are assumed, BAR can accommodate 1000 users capable of communicating with each other, with the same latency.

In the Cocaine Auction Protocol [24], an efficient anonymous broadcast technique is proposed. Communication is through an Ethernet or wireless medium. This protocol is intended for local area networking environments and is not practical for large-scale communications systems. Although it provides full anonymity, it is susceptible to side channel attacks.

The Buses protocol [1] is based on “bus-like” fixed message delivery routes that can be configured for different levels of anonymity. Different parameter settings can define different tradeoffs between time and communication complexity. In the original Buses protocol, the size of the bus, as well as the need to wait for all seats (messages) to be completed before a bus starts its route, makes the system not suitable for scalable, real-time communications.

In [16,17], variations of practical buses protocols have been proposed. In [17], a practical version of the buses protocol is used that maintains a fixed number of seats per user in each bus. The protocol still suffers from scalability issues, as the authors point out. The Taxis protocol variation [16] is more scalable in the cost of reducing sender anonymity to smaller sets.

The Drunk Motorcyclist Protocol [29] is a protocol providing strong anonymity properties, against a global active adversary. Although most of the other works cannot deal with active adversaries, the protocol of [29] is only suitable for connectionless anonymous messaging and not for real-time communications.

P^5 [22] is a system that aims to provide sender, receiver and sender–receiver anonymity against a global passive adversary. The protocol assumes a global broadcast channel, logically modeled as a broadcast tree. All participants send fixed length messages at a fixed rate (either real or noise). If users actually communicate they send the message encrypted with the public key of the receiver and every user in a broadcast channel will attempt to decrypt all received packets. P^5 uses bitmasks to balance broadcast costs versus anonymity: users exchange packets only if they use an appropriate bitmask. Although P^5 is scalable enough to support a large number of users, it requires a root broadcast channel and thus suffers from packet loss, if the number of active users exceeds a certain threshold.

Our work also uses broadcast channels and is conceptually close to [22]. However BAR is computationally more efficient, by allowing users to filter out noise traffic and actually decrypt only the packets destined for them. Moreover the use of a mix-net-based bridge mechanism removes the requirement for a root broadcast channel, making BAR practical for any number of users, provided that sufficient broadcast servers are added. By using a selective broadcast mechanism, BAR has lower end-to-end latency than P^5, for a much lower bandwidth. For example in [22] for 16 Kbps send rate without packet losses, users can gain anonymity for $N = 100$ by spending 2 Mbps of bandwidth, while in BAR for $N = 100$ and $q = 0.25$, users only need 0.4 Mbps bandwidth for an average connection latency 1.3 s.

6 Conclusion and future work

We have proposed an efficient, secure and scalable system for anonymous Internet communication that provides strong anonymity with forward secrecy against a global eavesdropper even when there is communication failure. The efficiency analysis shows that for N-anonymity, we can reduce the bandwidth to a fraction qN of N by using selective broadcast for a small increase in end-to-end communication latency.

The system is scalable and supports anonymous communication for any number of users assigned to different clusters, using the public session keys of the built-in key management mechanism. In future work, we plan to investigate the full distribution of the role of the BAR servers among the users and to extend our implementation to provide a large-scale system that captures the functionality of all the sub-protocols.

Acknowledgements The work of the first author was partially supported by the European Commission under the Horizon 2020 Programme (H2020), as part of the OPERANDO project (Grant Agreement no. 653704)

References

1. Beimel, Dolev: Buses for anonymous message delivery. J. Cryptol. **16**(1), 25–39 (2003)
2. Borisov, N., Danezis, G., Mittal, P., Tabriz, P.: Denial of service or denial of security? In: Proceedings of the 14th ACM Conference on Computer and Communications security, pp. 92–102. ACM (2007)
3. Burmester, M., Munilla, J.: Lightweight rfid authentication with forward and backward security. ACM Trans. Inf. Syst. Secur. (TISSEC) **14**(1), 11 (2011)
4. Chaum, D.: Untraceable electronic mail, return addresses, and digital pseudonyms. Commun. ACM **24**(2), 84–90 (1981)
5. Chaum, D.: The dining cryptographers problem: unconditional sender and recipient untraceability. J. Cryptol. **1**(1), 65–75 (1988)
6. Chen, C., Asoni, D.E., Barrera, D., Danezis, G., Perrig, A.: HORNET: high-speed onion routing at the network layer. CoRR abs/1507.05724 (2015). http://arxiv.org/abs/1507.05724
7. Corrigan-Gibbs, H., Ford, B.: Dissent: Accountable anonymous group messaging. In: Proceedings of the 17th ACM Conference on Computer and Communications Security. CCS '10, pp. 340–350. ACM, New York, NY, USA (2010)
8. Corrigan-Gibbs, H., Wolinsky, D.I., Ford, B.: Dining in the sunshine: verifiable anonymous communication with verdict. CoRR abs/1209.4819 (2012). http://arxiv.org/abs/1209.4819
9. Danezis, G., Clayton, R.: Route fingerprinting in anonymous communications. In: Peer-to-Peer Computing, pp. 69–72. IEEE Computer Society (2006)
10. Danezis, G., Dingledine, R., Mathewson, N.: Mixminion: design of a type iii anonymous remailer protocol. In: IEEE Security and Privacy Symposium (2003)
11. Dingledine, R., Mathewson, N., Syverson, P.: Tor: The second-generation onion router. In: Proceedings of the 13th USENIX Security Symposium (2004)
12. Dolev, S., Ostrovsky, R.: Xor-trees for efficient anonymous multicast receiption. In: Advances in Cryptology—CRYPTO'97 (1997)
13. Freedman, M.J., Morris, R.: Tarzan: a peer-to-peer anonymizing network layer. In: Proceedings of the 9th ACM Conference on Computer and Communications Security (CCS 2002) (2002)
14. Goldschlag, D.M., Reed, M.G., Syverson, P.F.: Onion routing for anonymous and private internet connections. Commun. ACM **42**(2), 39–41 (1999)
15. Günther, C.G.: An identity-based key-exchange protocol. In: Advances in CryptologyEurocrypt89, pp. 29–37. Springer (1990)
16. Hirt, A., Jacobson, M., Williamson, C.: Taxis: scalable strong anonymous communication. In: Modeling, Analysis and Simulation of Computers and Telecommunication Systems, 2008. MASCOTS 2008. IEEE International Symposium on, pp. 1–10. IEEE (2008)
17. Hirt, A., Jacobson Jr, M.J., Williamson, C.L.: A practical buses protocol for anonymous internet communication. In: Proceedings of the Third Annual Conference on Privacy, Security and Trust (PST2005), St. Andrews, New Brunswick, Canada, 12–14 Oct 2005(2005)
18. McLachlan, J., Tran, A., Hopper, N., Kim, Y.: Scalable onion routing with torsk. In: Proceedings of the 16th ACM Conference on Computer and Communications Security, CCS '09, pp. 590–599. ACM (2009)
19. Reiter, M.K., Rubin, A.D.: Crowds: anonymity for web transactions. ACM Trans. Inf. Syst. Secur. **1**(1), 66–92 (1998)
20. Ren, J., Wu, J.: Survey on anonymous communications in computer networks. Comput. Commun. **33**(4), 420–431 (2010)
21. Rennhard, M., Plattner, B.: Practical anonymity for the masses with morphmix. In: Juels, A. (ed.) Proceedings of Financial Cryptography (FC '04), pp. 233–250. Springer-Verlag, LNCS 3110 (2004)
22. Sherwood, R., Bhattacharjee, B., Srinivasan, A.: P5: a protocol for anonymous communications. J. Comput. Secur. IOS Press **13**(6), 839–876 (2005)
23. Shields, C., Levine, B.N.: A protocol for anonymous communication over the internet. In: Proceedings of the 7th ACM Conference on Computer and Communications Security (CCS-00), pp. 33–42. ACM, New York (2000)
24. Stajano, F., Anderson, R.J.: The cocaine auction protocol: On the power of anonymous broadcast. In: Pfitzmann, A. (ed.) Information Hiding, Lecture Notes in Computer Science, vol. 1768, pp. 434–447. Springer (1999)
25. Stoica, I., Morris, R., Liben-Nowell, D., Karger, D., Kaashoek, M., Dabek, F., Balakrishnan, H.: Chord: a scalable peer-to-peer lookup protocol for internet applications. IEEE/ACM Trans. Netw. **11**(1), 17–32 (2003)
26. Tabriz, P., Borisov, N.: Breaking the collusion detection mechanism of morphmix. In: Proceedings of the 6th International Conference on Privacy Enhancing Technologies. PET'06, pp. 368–383. Springer-Verlag, Berlin, Heidelberg (2006)
27. The anonymizer. https://www.anonymizer.com/
28. Warren, J.: Bitmessage: A peer-to-peer message authentication and delivery system. white paper (27 Nov 2012), https://bitmessage.org/bitmessage.pdf (2012)
29. Young, A.L., Yung, M.: The drunk motorcyclist protocol for anonymous communication. In: Communications and Network Security (CNS), 2014 IEEE Conference on, pp. 157–165. IEEE (2014)