Incentive Human-to-Human Chat Support Protocol

Khalid Alhamed and **Marius C. Silaghi**
Florida Tech

Abstract

In this study, we introduce a new peer-to-peer (P2P) approach to instant messaging systems based on a fully decentralized network, and where each human owning a peer can control the traffic supported by her system. The control may be based on criteria such as: (a) her desire to help the endpoints of the communication, e.g., based on friendship, (b) her desire to help a cause, e.g., based on the content/topic of the communication, (c) reputation, or (d) the utility brought to her by the handled data. Peers need help to communicate when they are behind NATs. Unlike the P2P chat system used in Skype, no centralized servers are involved (no central redirect server, login server or web server).

Providing intrinsic motivation such that peers help with traffic is important in order to eventually make an open source P2P chat system viable. In current non-incentive P2P systems like Skype, availability of open source versions can potentially starve the system of supernodes (since users can disable the resource consuming supernode-function).

In our approach, each peer has equal privileges to any other peer. Nodes register their address with other peers of their choice, which can then act as directories on their behalf. Each peer with sufficient resources can voluntarily play the role of directory for her friends or of forwarding incoming messages to peers behind NATs.

This paper analyzes key functions of the solution such as incentive management, NAT and firewall traversal, connection establishment and message transfer under different network setups.

Introduction

Instant messaging has been one of the most successful applications on the Internet, starting with `talk` and `IRC` to `Twitter` and `Facebook`. One of the most currently used such systems, Skype, is a peer-to-peer (P2P) application. The P2P architecture offers Skype significant robustness and scalability, as well as efficiency, particularly from the perspective of a low latency.

Besides depending on a centralized set of login servers that could be used to eventually bring down the whole system, Skype is a closed source software, and it is known to use computational resources of unsuspecting users.

While efforts to build open source clients have failed due to secret changes in protocols (Baset & Schulzrinne 2004; 2006; Guha, Daswani, & Jain 2006), it is questionable whether they could in fact survive when the protocol would be open. The issue is that, with open source software, the users can get versions that disable expensive super-node functions that exploit the resources of the nodes. This missing property is called faithfulness (Shneidman, Parkes, & Massoulié 2004). In the closed source chat systems, the only incentive to act as supernode is the fact that binaries do not allow users to disable that function selectively. Since current protocols have no intrinsic incentives to voluntarily act as a supernode, availability of an open source agent can starve the network of super-nodes and lead to the demise of the service.

We propose an approach to P2P supernodes that can simultaneously offer advantages similar to Skype, while being fully decentralized, open source, allowing customization and control of one's resources. Users may thereby enforce that their resources are used only for causes that they want to support.

Providing intrinsic incentives for peers to help with traffic flow in P2P network applications is important in order to overcome the most common problems with the open-source P2P paradigm: Free-riding and tragedy of the commons (Ma *et al.* 2003). Incentives are intrinsic to an application if the participant loses by not performing the actions expected of her due to those actions, and not due to benefits provided separately (e.g., bundled in a pre-compiled software).

Incentives To encourage peers to support the chat communication of others, the proposed protocol enables incentives, such as:

- Ability to control the traffic passing by her system:
  - helping the endpoints of the communication (e.g., based on friendship or subject).
  - rejecting the endpoints of the communication (e.g., based on disagreement).
- Getting information concerning who talks to whom and what are they talking about (for research, marketing data or pure curiosity).
- Supporting open source P2P chat systems, thereby supporting the freedom of customization.
- Opportunity to insert advertisements into the stream.

- Ability to offer paid services for supporting the communication.

The characteristics of our P2P approach is: network address translation (NAT) piercing ability, democracy, no outside control, full decentralization. In the next section we discuss differences with related work. After introducing the main concepts, section Protocol describes in detail the mechanism used by peers for incentive chat.

Background

Skype (Baset & Schulzrinne 2004) is a P2P application based on the Kazaa architecture for voice over IP (VoIP) and instant messaging (IM). It offers multiple services, such as: (a) VoIP allows two Skype users to establish two-way audio streams with each other and supports conferences of multiple users, (b) IM allows two or more Skype users to exchange small text messages in real-time, and (c) file-transfer allows a Skype user to send a file to another Skype user.

The implementation used by Skype is not open source and its protocol is not publicly disclosed. In addition, it is not fully decentralized, relying on a set of pre-established nodes for login authentication and as last solution to search users registered or logged into the Skype network (Baset & Schulzrinne 2004). Skype lacks faithfulness (Shneidman, Parkes, & Massoulié 2004), in the sense that users will benefit by disabling their super-node functions, thereby bringing down the system. A node should have a public IP address in order to become a supernode. Skype has solved the problem of unreachable nodes behind NATs or firewalls by implementing a version of the STUN and TURN protocols in each node (Rosenberg *et al.* 2008; Mahy, Matthews, & Rosenberg 2010; Rosenberg 2003). If one client is behind a NAT, Skype uses connection reversal whereby the node behind the NAT initiates the TCP/UDP media session regardless of which end requested the VoIP or file-transfer session. If both clients are behind NATs, Skype uses a STUN-like NAT traversal to establish the direct connection. In the event that the direct connection fails, Skype falls back to a TURNlike approach where the media session is relayed by a publicly reachable supernode (Guha, Daswani, & Jain 2006).

A P2P Java platform called JXTA was released in 2001 and is widely used for a variety of P2P applications (Gradecki 2002). It can be relatively easily configured for communication within groups defines as subsets of a unique group structure in charge of managing their identifiers. It also provides the needed NAT support. Various flavors of P2P platforms for social networks are provided as both open source and close source applications: Bittorrent, PeerSon (Buchegger *et al.* 2009; Cohen 2003).

Concepts

A *NAT* is a mechanism to share a network connection with a unique IP from several machines on a intranet. Each socket on a machine on the intranet is viewed from the outside as having the same IP, but potentially a different port than the one actually used on its own machine.

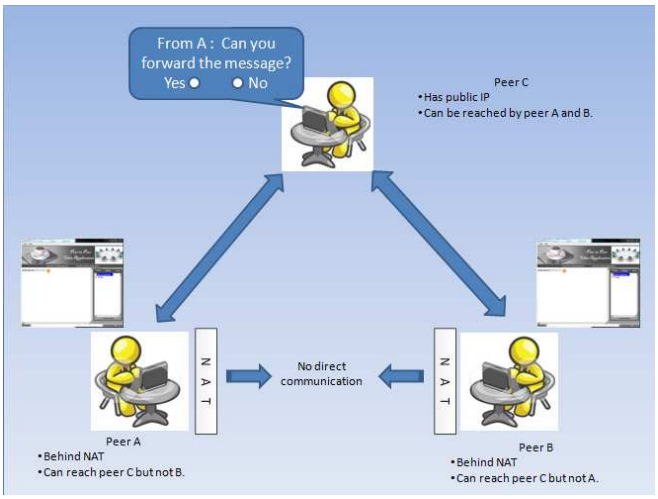

Figure 1: Providing incentives for relaying messages

There are several types of NATs. With NATs, when both peers are found behind NATs and one of the NATs is symmetric, then all the communication has to be performed using an external server (e.g., with the TURN protocol). With full-cone NATs, two devices found both behind such NATs can communicate directly if their communication is initialized by an external server (e.g., with the STUN protocol).

Definition 1 (Supernode) *A supernode is an agent that is not found behind a NAT or firewall, and has a public IP.*

Definition 2 (Willing Forwarding) *A supernode is* willingly forwarding *a chat request if the owner of the supernode has specifically agreed with the corresponding action.*

Definition 3 (Directory Server) *An agent that accepts to keep track of the IP of a peer and to relay its current address when queried.*

An agent can simultaneously be a directory server and a supernode. Unlike DNS, in our case we do not assume directory servers to be organized hierarchically, and the mechanism peers learn one's directory servers is outside the scope of this report.

Definition 4 (Address Container) *An address container is an object (e.g., a file) that contains information about a peer, such as:*

- *the name of the peer*
- *the certificate of the peer, usable to verify its identity on connection*
- *a list of common socket addresses (IP and port) where the peer can be usually contacted*
- *a list of directory servers supporting this peer with connections when he is not at one of the common socket addresses*

Provided incentives

Our study assumes rational behavior, where users of different types can have utilities that depend on various incentives.

The incenteves we address are categorized into the following types:

1. **Curiosity:** e.g. getting information about who talks what to whom.
2. **Commercial perspective:** e.g. opportunity for paid services or inserting advertisements.
3. **Altruism:** e.g. supporting unknown or known others to freely communicate.

For *Full-Cone NATs*, once the communication is launched, the supernode can be bypassed as soon as the peers can exchange their Internet addresses. In this case the incentive that a supernode can get consists of its ability to learn that users do communicate. To reduce the chance that it provides support to a cause it opposes, or just for gathering data, the supernode can request a declared topic for each connection request. E.g., if the declared topic matches items on a black-list, the requesting user can be denied.

With *Symmetric NATs* all the communication has to pass via a relaying supernode (Figure 1). The load of this supernode is significant since it has to transmit the whole data. An *incentive* automatically available to a supernode based on open source is that it can *learn the amount of communication* between peers, knowledge that can provide motivation to some researchers, marketing departments, etc. Other supernodes put other conditions (other than payments):

- being allowed to insert ads in the stream of data. While peers may insert digital signatures on blocks of data to be able to automatically remove such ads, supernodes may request dynamic interaction to verify that peers have read the ads (such as CAPTCHA: ask receivers to answer a question based on the ads they were supposed to see).
- being given access to communication in plaintext (a request that data not be encrypted). While some people may still exchange secret data used using steganography, supernodes can use public data in the communication for marketing, studies, curiosity, etc.

Terms The parameter $terms$ sent with negotiation messages consists of a structure of the type $OR(term_1, term_2, ...)$, i.e., a disjunction of choices, where each $term_i$ is a pair of the form $(type, requests)$ where $type$ specifies the communication mechanism possible with the requested peer (direct, forwarded: see the four cases in the next section), while $requests$ is a tuple $\langle topic, plaintext, ads, payment\rangle$, each of these elements being a Boolean specifying whether the corresponding feature is requested or not.

Protocols

In this section we introduce the protocol followed by each participant. First we give the perspective of the user. A user follows the sequence of interactions below:

1. Download **container** of address from destination website/email into her agent. This has to be done only once in the lifetime of the relation between the two peers.
2. For an existing address, try direct connections first, if available. If they work, go to step 5.
3. Send **request** message (`Help`) to supernodes, in parallel. Available supernodes answer with **negotiation** messages (`Welcome`).
4. Choose supernode S with best terms and send **agree** message (`Confirmation`).
5. Send **data** message based on the obtained procedure.

In the scheme above, at the first step the agent will save the address in the local storage for potential further usage.

The protocol of a supernode is given in Algorithm 1. The supernode uses as data structure a hashtable $channel$ with names as keys and addresses as values. Supernodes handle the **request** messages with the procedure *ForwardRequest*. The **agree** messages are handled with the procedure *AgreeForwarding*, and the **data** messages are handled with the procedure *ForwardingData*. Note that these procedures enable communication behind symmetric NATs, but this is sufficient to start any other type of communication possible, once peers can communicate with each other and if their communication is not censored. Nevertheless, for direct communication from behind two full-cone NATs they additionally need the cooperation of the supernode in providing them with their external addresses.

```
when ForwardRequest(src_IP, source, destination)
    authenticate(src_IP, source);
    if not willingly_forwarding(source, destination) then
        return;
    terms → myterms[source];
    send negotiation(terms) to src_IP;
when AgreeForwarding(src_IP, source, destination, terms)
    check terms ;
    if NAT_opening(terms) then
        send (src_IP) to destination;
    else
        open channel[destination] adding source to set of
        its tags;
        open channel[source] adding source to set of its
        tags;
when (ForwardingData(source, destination, message))
    if (!check_forwarding(source, destination) then
        return
    send data message to channel[destination];
```

Algorithm 1: Supernode protocol

In order to understand the communication initialization between peers, we illustrate four cases that could appear when a peer tries to connect to another peer (which mirrors related cases of the STUN protocol, enriched with handling of incentives). For clarity, we first present the four cases with a potentially inefficient version of the initialization, which increases the latency by one round-trip. Subsequently we show that the latency of this initialization can be reduced in certain circumstances to the latency for the case without incentives (STUN):

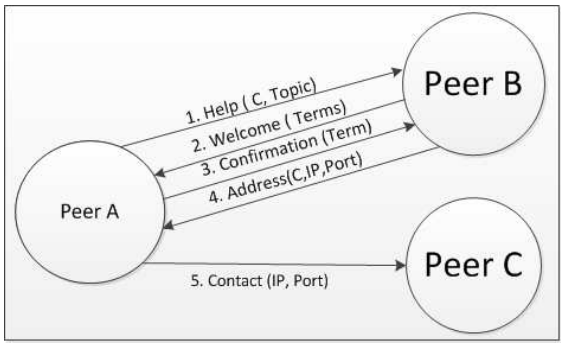

Figure 2: [Case 1] Peer A can reach peer B but does not know the IP of peer C

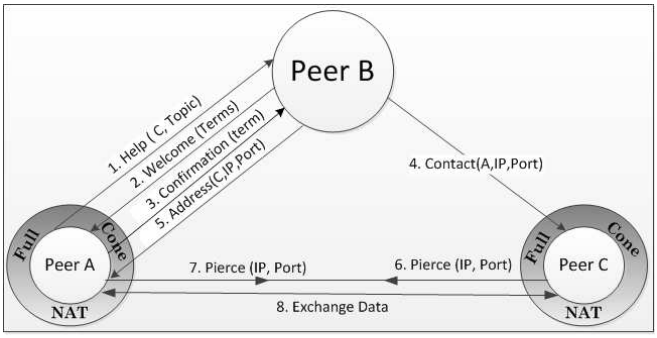

Figure 3: [Case 2] Peer A and B are behind full-cone NATs. They can reach peer B but they cannot reach each other

Case Roaming Peer (No NAT) In this case peer_A is behind NAT and she wants to send a message to peer_C, but peer_A does not have the current address of peer_C (see Figure 2). On the other hand, peer_B has peer_C's address stored in its directory server. Therefore, peer_A needs peer_B's help in order to communicate with peer_C. Both peer_B and peer_C have public IP addresses.

1. Peer_A sends a help request to peer_B. The request has two parameters (1) destination information in this case it is peer_C public identification (PID) and optionally (2) topic information (why does peer_A want to communicate with peer_C).
2. Then, peer_B sends a welcome message to peer_A including, peer_B's terms to help. As aforementioned, terms include: access to plaintext messages, insertion of ads, forward only (address of peer is not offered due to symmetric NAT or to preclude bypassing).
3. Peer_A sends an acceptance confirmation message to peer_B.
4. Peer_B sends a message to peer_A containing the contact information of peer_C (IP address and port).
5. Finally, peer_A can connect to peer_C.

Case Full-Cone NATs In this case both peer_A and C are behind a full-cone NATs (see Figure 3). They can reach peer_B but they cannot reach each other. Peer_B has a communication channel with peer_C. Therefore, peer_A needs peer_B's help in order to communicate with Peer_C.

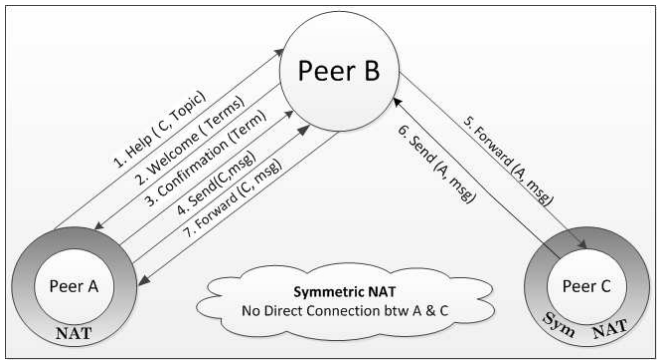

Figure 4: [Case 3] Peer A and C are behind NATs, at least one of the NATs being symmetric. They can reach peer B but they cannot reach each other

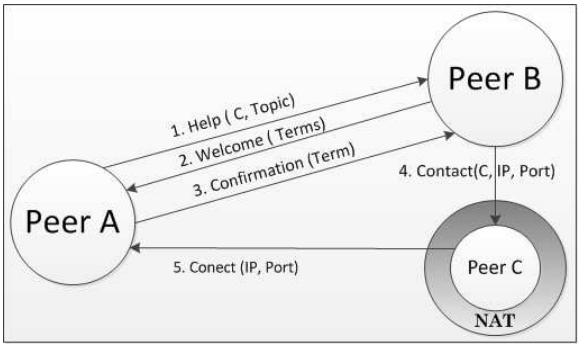

Figure 5: [Case 4] Peer C is behind the NAT. Peer A can reach peer B but cannot reach peer C

1. Peer_A sends a help request to peer_B.
2. Then, peer_B sends a welcome message to peer_A including, peer_B's terms for help.
3. Peer_A sends an acceptance confirmation message to peer_B.
4. Peer_B sends a contact message to peer_C containing the contact request information of peer_A (IP address and port).
5. Peer_B sends an address message to peer_A containing the contact information of peer_C (IP address and port).
6. Peer_A and peer_C pierce their NATs.
7. Finally, peer_A and peer_C have a communication channel to exchange messages.

Case Symmetric NAT In case both peer_A and peer_C are behind a NAT and one of the two NATs is symmetric (pictured at peer_C in Figure 4), then forwarding between them cannot be avoided.

1. Peer_A sends the help request to peer_B.

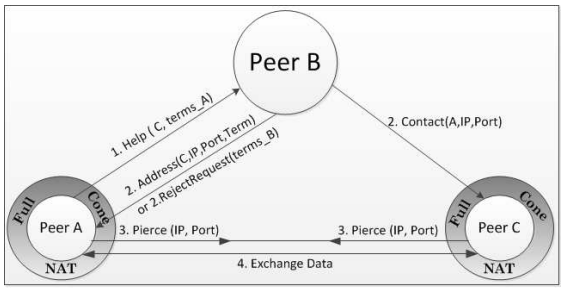

Figure 6: Reducing latency

2. Peer_B sends a welcome message to peer_A including, peer_B's terms to help.
3. Peer_A sends an acceptance confirmation message to peer_B.
4. Peer_A sends a message to peer_B containing a message.
5. Peer_B forwards the message to peer_C with the data of the sender and potentially with additional processing (adds insertion, extra negotiation with peer_C).
6. Answers from peer_C follow the same scheme.

Case NAT at Destination The case when the initiator peer_A is not behind a NAT but peer_C is in this situation and has a communication channel open with peer_B, then the procedure is similar to the one at the first case.

1. Peer_A sends a help request to peer_B.
2. Peer_B sends a welcome message to peer_A including, peer_B's terms to help.
3. Peer_A sends an acceptance confirmation message to peer_B.
4. Peer_B sends a message to peer_C containing the contact information of peer_A (IP address and port).
5. Finally, peer_C can connect to peer_A.

Reducing Latency of Initialization The previously described simplified handshake mechanism has a higher latency than STUN due to the extra round-trip used to agree on the terms for service.

This handshake can be brought to the latency of STUN whenever the client accepts to proactively offer the services that the supernode requests. The `Help` message is allowed to optionally contain the list of `terms` that the client is ready to accept (willing to do list). Instead of exchanging three messages between peers for terms agreement: `Help`, `Welcome` and `Confirmation`, the client only needs to send a single message: `Help`. In case the supernode finds acceptable `terms` among the `terms` offered by the client, then it returns the most preferred among them, `term` to peer_A using an `Accept` message. The example in Figure 6 details the scenario of full-cone NATs, but it applies identically to the other discussed cases. If the list of terms does not match its acceptable terms then it can reply with a `RejectRequest` message specifying its terms. The client can eventually retry the communication request including some of the requested terms.

The peer is configured by a user with a list of acceptable terms tagged with a preference value: $\langle(term_1, p_1), (term_2, p_2), ...\rangle$. First the software agent may try to negotiate the terms $term_1$ with the highest preference p_1 using known supernodes serving peer_C. If that fails then it can try $terms_2$ with the lower preference value p_2, until it exhausts its acceptable terms. The user is presented with the terms suggested by supernodes in their rejection replies. Users can change their default terms for certain supernodes, based on answers to their past requests, to reduce the latency of subsequent connections. The agent can automatically infer based on recent past answers whether a query will fail and optimize query of supernodes.

Addressed attacks

- Fake service claim: The expected behavior of a supernode is to provide communication services to registered peers based on a settled terms agreement. However, some supernodes may take advantage of this function in different ways. Some attacker may claim to provide the supernode function (forwarding messages, proving addresses or startup communication) just for gathering start-up data, without actually providing services. It is up to the peer using it as a directory to detect and protect herself from these attacks, and a recommendation system can be build to evaluate supernodes.
- False identity declarations: A mechanism to verify the identity of a user in DDP2P is described in (Qin *et al.* 2013; Silaghi *et al.* 2013). Intuitively, some identities are verified directly using a protocol based on SMS and previously-known email addresses of friends (Silaghi *et al.* 2013). Remaining identities are verified indirectly using witness stances in a authoritarian or bottom-up census process (Qin *et al.* 2013).
- Supernode evasion: a user could indeed ask the address of an intermediary as a way of hiding the final destination. This approach is similar to the attack mentioned in the article where the peers use steganography for exchanging text privately over a supervised channel. These techniques are common even now with existing email infrastructures (see anonymous remailers), and we expect that they will happen with our system, too. It already impacts trust in claims of NGOs as to the real purpose of their activities, but it does not lead to a lack of survivability of NGOs. This potential attacks somewhat reduce the incentives of type one (curiosity), and somewhat more the incentives of type three (serving selectively certain causes). They have less impact on incentives of type two.

Conclusions

We address the problem of providing incentives for users to let their systems serve as supernodes (helpers for initiating or forwarding communication) in an open source P2P chat

protocol. We remark that one of the common P2P applications for chat, Skype, does not provide faithfulness. Namely, the only reason for which certain users allow their Skype agent to relay messages is that they cannot turn it off (given that it is a close source software).

The research question addressed here is how to provide intrinsic incentives for volunteers to serve as supernodes, in a way that would make an open source chat protocol viable. The observation is that human volunteers have several ways in which they can benefit from helping with establishing connections between others. Some benefits come from common characteristics of humans.

A human benefit for volunteering to send data is the good feeling of *serving a noble cause*. While some volunteers would simply offer their resources for the cause of *open source*, others may want to learn a declared *topic* of the discussion, or to even request access to the whole communication (in plaintext), to guarantee that they are happy to support the *cause* of the given connection. This may be acceptable to certain users and for certain communications.

Another human characteristic that can motivate a volunteer is *curiosity* of who talks to whom, of how much they talk and, if public, on what topics. This is data that people often offer willingly or unwillingly to close software applications, but in our case users themselves gain by knowing how much information they actually give away.

A third intrinsic benefit of the supernodes can be commercial. The collected data can be used for marketing, studies, etc. Supernode users can also get credit for a *quid pro quo* help if in the future they will be behind a NAT. Moreover, in some models (e.g., with symmetric NATs) the supernodes that have to forward the whole data can be enabled to insert advertisements into the stream of data. Just as in the previous cases, the same information is currently leaked to closed source software, while the advantage of open source is to let the user know exactly what privacy they are losing. Based on volunteer supernodes we now establish an incentive fully decentralized open source system for chat.

References

Baset, S. A., and Schulzrinne, H. G. 2004. An analysis of the skype peer-to-peer internet telephony protocol. In *TR CUCS-039-04*.

Baset, S. A., and Schulzrinne, H. G. 2006. An analysis of the skype peer-to-peer internet telephony protocol. In *INFOCOM*.

Buchegger, S.; Schiöberg, D.; Vu, L. H.; and Datta, A. 2009. PeerSoN: P2P social networking. In *WSNS*, 46–52.

Cohen, B. 2003. Incentives build robustness in bittorrent. In *Workshop on Economics of Peer-to-Peer systems*, volume 6, 68–72.

Gradecki, J. 2002. *Mastering JXTA*. Wiley.

Guha, S.; Daswani, N.; and Jain, R. 2006. An experimental study of the skype peer-to-peer voip system. In *IPTPS 2006*.

Ma, R. T. B.; Lee, C. M.; Lui, J. C. S.; and Yau, D. K. 2003. Incentive p2p networks: A protocol to encourage information sharing and contribution. *SIGMETRICS Performance Evaluation Review* 31.

Mahy, R.; Matthews, P.; and Rosenberg, J. 2010. Traversal using relays around nat (turn). IETF, RFC 5766.

Qin, S.; Silaghi, M.; Matsui, T.; Yokoo, M.; and Hirayama, K. 2013. P2p decentralized population census. In *Workshop on Decentralized Coordination*.

Rosenberg, J.; Mahy, R.; Matthews, P.; and Wing, D. 2008. Session traversal utilities for nat (stun). IETF RFC 3489.

Rosenberg, J. 2003. Interactive connectivity establishment (ICE): a methodology for network address translation (NAT) traversal for the session initiation protocol (SIP). IETF.

Shneidman, J.; Parkes, D.; and Massoulié, L. 2004. Faithfulness in internet algorithms. In *SIGCOMM*.

Silaghi, M.; Qin, S.; Alhamed, K.; Matsui, T.; Yokoo, M.; and Hirayama, K. 2013. P2p petition drives and deliberation of shareholders. In *Workshop on Decentralized Coordination*.