

Hidden Communication in P2P Networks: Steganographic Handshake and Broadcast

Raphael Eidenbenz
ETH Zurich, Switzerland
eidenbenz@tik.ee.ethz.ch

Thomas Locher
IBM Research Zurich, Switzerland
thl@zurich.ibm.com

Roger Wattenhofer
ETH Zurich, Switzerland
wattenhofer@tik.ee.ethz.ch

Abstract—We consider the question of how a conspiring subgroup of peers in a p2p network can find each other and communicate without provoking suspicion among regular peers or an authority that monitors the network. In particular, we look at the problem of how a conspirer can broadcast a message secretly to all fellow conspirers. As a subproblem of independent interest, we study the problem of how a conspirer can safely determine a connected peer’s type, i.e., learning whether the connected peer is a conspirer or a regular peer without giving away its own type in the latter case. For several levels of monitoring, we propose distributed and efficient algorithms that transmit hidden information by varying the block request sequence meaningfully. We find that a p2p protocol offers several steganographic channels through which hidden information can be transmitted, and p2p networks are susceptible to hidden communication even if they are completely monitored.

I. INTRODUCTION

The reader may be familiar with the archetypal espionage scene where an agent meets his contact person for the first time. In order to make sure that the agent got the right person, and not some innocent bystander, they exchange previously agreed-upon pass phrases.¹ In this paper we study this spy rendezvous problem in the context of computer networks. One natural habitat are peer-to-peer (p2p) or overlay networks, where conspiring peers (“spies”) strive to find each other and exchange secret messages, unbeknownst to the “regular” peers in the networks. Such protocols can be useful in different contexts. In p2p networks, e.g., conspiring peers may prioritize each other in terms of quality of service. In overlay networks, conspiring peers may try to position themselves at strategically favorable spots to manipulate the overlay. Conspiring peers may also be machines controlled by law enforcement in order to bring down a malicious botnet. As can be seen from these examples, conspiring peers may be considered useful or harmful, depending on the point of view. For the sake of a lucid presentation, we describe the situation from the point of view of the conspiring peers.

Depending on the application, conspiring peers may need different communication primitives. In this paper, we focus on the so-called *broadcast problem* where a conspiring peer wants to send a secret message to all other conspiring peers, either directly or indirectly, however, without getting caught by the regular peers.

¹Indeed, such spy rendezvous protocols are not an invention of literature or film. For example, it is said that atomic spy Klaus Fuchs hooked up with his contact person in New York by carrying a tennis ball, using the pass phrase “Can you tell me the way to Grand Central Station?”

II. MODEL & PROBLEM DEFINITION

We are given a p2p network, which consists of a set P of $|P| = n$ peers. Each peer $u \in P$ can communicate directly with any other peer v if u has previously learnt the address of v . It is assumed that a peer does not know any other peers initially, i.e., upon joining the network a peer is not connected to any other peers. In order to establish connections to other peers, we assume that there is a publicly known server that is aware of all peers currently in the network. Upon request this server delivers a list of k peer addresses, which are chosen uniformly at random from all peers. If two peers are connected, we say that they are *neighbors*. Another common technique in popular p2p networks is to inquire known peers about other peers in the network. Since this approach still depends on some kind of bootstrapping mechanism, we assume for the sake of simplicity that peer addresses are only provided by the dedicated server. Furthermore, all communication is assumed to be reliable, i.e., message loss and corruption can be handled in a canonical way using redundancy and/or checksums. However, message delivery times are subject to a potentially variable delay.

In general, a p2p network may be used to exchange any number of files. We assume that there is only one file f that is shared in the network, i.e., our definition of a network is akin to the concept of a BitTorrent *swarm* in which all peers share a single file or a specific collection of files. Efficient dissemination of the file f is achieved by splitting it into m data blocks $b_1, b_2, \dots, b_m$, and trading locally available blocks for missing blocks with other peers, which is common practice in most file sharing networks. Both the number of blocks m and the number of peers n are assumed to be fairly large and in the same order of magnitude so that $m \gg \log n$ and $n \gg \log m$.² We assume that the file blocks are fairly well distributed among all peers and at least one peer holds all m blocks at any time. Thus, it is always possible to acquire the entire file f by connecting to a reasonable number of peers. Furthermore, we make the assumption that any two connected peers regularly exchange information about the locally available blocks, i.e., a peer reports regularly on the blocks it has to each neighbor. Over each link, a peer can send one request for a block b_i at a time to a certain peer, wait until b_i has been transmitted completely, and then send the next request to the same peer.³ Unless a block is already being

²The base of the logarithm is always 2 unless we write $\ln$, in which case the base is e .

³Note that it is possible to send different requests to different peers simultaneously.

transmitted, a request is always served and the requesting peer receives the block at the latest after a certain bounded delay.

We distinguish between two classes of peers: A peer u is either a *regular peer* or a *conspirer*. A regular peer is a peer whose sole purpose is to acquire and share file f with other peers. The conspirers, on the other hand, have a secret agenda. In particular, the conspirers strive to secretly communicate among each other. The set of conspirers is denoted by C , and its cardinality is $|C| = c$. Another distinction between regular peers and conspirers is that the conspirers share an exclusive secret K of length $|K|$. What is more, the conspirers know both the number c of conspirers and, for ease of presentation, the number n of peers in the network.⁴

In order to ensure that the p2p network is used only for the intended purpose of sharing file f and to prevent fraudulent behavior by conspiring peers, there is an authority that monitors the network. If the authority ever learns that a peer u does not abide by the rules of the imposed protocol and exchanges other information with certain peers, then u is punished, e.g., by adding u to a globally available (signed) blacklist or, if the authority has the power, by expelling u directly from the network. The authority can detect conspirers in two ways, either it observes suspicious communication directly, or regular peers denunciate them, i.e., we assume that there is an incentive for regular peers to report any observed departure from the file sharing protocol. Of course, it is also possible for conspirers to report regular peers but we assume that it is not worthwhile for conspirers to do so for the following reasons. First, the authority may suspect both the defendant and the accuser, which may be detrimental to the conspirer as well. Second, assuming that n is considerably larger than c , other regular peers may vouch for the accused (regular) peer and thereby revealing the disingenuous nature of the conspirers. In other words, c may not be large enough for the conspirers to campaign against a regular peer. Finally, the authority may have recorded the communication. In this case, it can determine that the regular peer always adhered to the protocol contrary to the accusation.

As mentioned before, the primary goal of the conspirers is to communicate with each other, while the authority tries to detect as many conspirers as possible. The main problem that we consider is called $BROADCAST(M)$, which is defined as follows.

$BROADCAST(M)$: A conspirer $u \in C$ wants to send a message M , directly or indirectly, to all other conspirers without raising suspicion among the regular peers and without being caught by the authority.

We are interested in how large a message M can be so that $BROADCAST(M)$ can still be solved with a high probability of success. Furthermore, the solution should be efficient in terms of *communication* and *space complexity*, i.e., the number of messages that must be exchanged as well as the number of bits that each conspirer has to store ought to be small.⁵

⁴As (BitTorrent) trackers usually provide only an estimate of n , we will argue in a later section that our techniques can easily be adapted to the scenario where the conspirers merely have an estimate of n and c .

⁵Apart from the downloaded blocks, a peer must also store, e.g., the addresses of its neighbors.

In order to solve $BROADCAST(M)$, we need to address the subproblem that a conspirer must be able to find out whether a particular neighbor is a regular peer or a conspirer without revealing that it is a conspirer itself. This problem, which we call $REVEALTYPE$, is of independent interest in itself.

III. HIDDEN BROADCAST IN P2P

The extent to which conspirers are able to communicate secretly among each other depends on the freedom that the imposed p2p protocol offers. If the peers are given more leeway in their actions, more information can be hidden. In our model, the order in which a peer requests blocks from a neighboring peer is not specified. Hence, the conspirers can introduce a logic to the order of request sequences and thereby communicate without violating the given protocol. We call such exploitable, variable parameters of the p2p protocol *steganographic channels*⁶. In the following, we will primarily make use of the *request order channel* that we have just described. Section III-F discusses additional steganographic channels in p2p networks that could be exploited as well. Obviously, a conspirer may also communicate freely with a neighboring conspirer if the network connection between them is not monitored; however, as the conspirers initially do not know the other conspirers' identities, they first have to determine their neighbors' types by means of a *steganographic handshake* (Section III-A). The conspirers' capabilities to communicate depend on the freedom they have in varying the imposed protocol. This freedom, in turn, is derived directly from the power of the authority monitoring the p2p network. In the subsequent sections, the goal is to determine how (much) information can be secretly exchanged given a certain authority. The discussion is structured according to increasing monitoring capabilities.

A. Steganographic Handshake

In this section, we describe our approach to solve $REVEALTYPE$, which is an essential building block for all proposed broadcast mechanisms. As mentioned before, we use the *request order channel* to exchange messages in secrecy. Let us assume for the moment that we have a mechanism to transmit bits over this channel, i.e., bits are transmitted by requesting blocks in a specific order. How many bits have to be exchanged in order to ensure that the communication partner is indeed a conspirer as well? The peer with the smaller id, say u , may send, e.g., the first half of the secret key K to v . If v is a conspirer, it knows that a conspirer tries to send the secret over the request order channel, and checks whether the bits produced by the received request sequence correspond to the first half of K . If this check is positive v knows that $u \in C$ and sends the second half of K back to u in plain text. Thus, u is ensured that v is a conspirer, too. On the other hand, if v is a regular peer, it does not notice any irregularity while communicating with u since any request order is allowed by the p2p protocol. Moreover, even if v was aware of the request

⁶Steganography is the art of hiding information in a message, such that only intended recipients are able to decipher the hidden information, and all other viewers of the message do not even suspect the existence of hidden information.

Algorithm 1 ENC_{order}

Input: block sequence B , message $M \leq |B|!$ **Output:** permuted block sequence Π

```

1: Sequence  $\Pi := \emptyset$ ;
2: for  $i := |B| - 1$  to 0 do
3:    $l := M \text{ div } i!$ ;
4:    $\Pi.append(B_l)$ ;
5:    $B.remove(l)$ ;
6:    $M := M - l \cdot i!$ ;
7: return  $\Pi$ ;
```

order channel and could decode the sent bits correctly, it would not be able to detect the irregularity since it does not know K . Note that also a regular peer (unknowingly) produces bits on the request order channel, and hence, v cannot distinguish u from a regular peer unless it knows K . The only problem is that a regular peer may accidentally send its requests in the same order as if it were a conspirer, i.e., it inadvertently sends the right $|K|/2$ bits over the steganographic channel, and a receiving conspirer would send a plain text message back to u , which is an illegal action. However, by choosing a key that is large enough and uniformly at random, the conspirers can keep the probability of this false positive fairly low: The probability of an individual false positive is $2^{-|K|/2}$. Hence, if the conspirers choose a key K of length at least $6 \log n$, then there are no false positive over any of the at most $\binom{n}{2}$ communication links *with high probability* (w.h.p.).⁷

The fact that a regular peer can have several neighboring conspirers poses a certain threat: A regular peer could perform a *replay attack*, i.e., it could request blocks in the same order as other peers requested them, and thus provoke a false positive with significant probability. The conspirers can avoid such an attack by using keys that are connection-specific. In particular, a conspirer u could use $\mathcal{H}(id_u || id_v || K, \log n)$ as a key when communicating with v , where “ $||$ ” is the concatenation operator and $\mathcal{H}(x, b)$ is a hash function mapping a bit string $x \in \{0, 1\}^*$ to a bitstring of length b with the property that if the input value x has an entropy of at least b then the hash value $\mathcal{H}(x, b)$ can be guessed successfully with probability at most 2^{-b} . Given that K is chosen uniformly at random, the input chosen by a conspirer has an entropy of $|K|$ bits. Hence, if $|K| \geq \log n$ then $\mathcal{H}(id_u || id_v || K, \log n)$ can be guessed by a regular peer with probability at most $1/n$. Again, in order to ensure that we get this probability over all communication channels, the length of the hash value should be increased to $3 \log n$, and K should consist of at least $3 \log n$ randomly chosen bits.

We will now discuss how the block request order can be used to exchange information. Once a conspirer u has determined which blocks it wants to request from a neighboring peer v it can permute the order of this request sequence as shown in Algorithm 1 to transmit $\lfloor \log(|B|!) \rfloor$ bits, where input B is the block sequence ordered according to the order in file f . Algorithm 1 interprets message M as a number, converts it

⁷If an event occurs with probability at least $1 - \mathcal{O}(1/n)$, we say that it occurs “with high probability”. A stronger definition demands that the probability is at least $1 - 1/n^{\Omega(1)}$, which can frequently be achieved by linearly increasing the constants involved (the constant in the exponent depends on the linear increase). For the sake of simplicity, we use the weaker definition throughout this paper.

Algorithm 2 DEC_{order}

Input: permutation Π **Output:** message $M \in \{0, 1\}^{\lfloor \log(|\Pi|!) \rfloor}$

```

1: Sequence  $S := (1, 2, \dots, |\Pi|)$ ;
2:  $M := 0$ ;
3: for  $i := 0$  to  $|\Pi| - 1$  do
4:    $l := S.index\text{Of}(\Pi_i)$ ;
5:    $S.remove(l)$ ;
6:    $M := M + l \cdot (|\Pi| - i - 1)!$ ;
7: return  $M$ ;
```

to its representation in the factorial number system, denoted by $M_!$, and computes a permutation Π by interpreting $M_!$ as the Lehmer code of Π . In the factorial number system representation, $x_!$, of a number $x \in \mathbb{N}$, the i -th digit has a place value of $i!$. As an example, the decimal number 17, which represents the binary message 10001_2 , has a factorial number representation of $2210_!$ because $0 \cdot 0! + 1 \cdot 1! + 2 \cdot 2! + 2 \cdot 3! = 17$. The Lehmer code of a permutation counts the number of swaps of neighboring elements that have to be executed in the originally ordered list for each element in the target permutation to be moved to its right position, starting from the first. The permutation $(3, 4, 2, 1)$, e.g., has a Lehmer code of $2210_!$ as, starting from the original sequence $(1, 2, 3, 4)$, element 3 needs two swaps to get to the left most position, then element 4 needs two swaps to get to the second position, element 2 can be moved to the third position with one swap, and element 1 is already at position four. To get a permutation from a given Lehmer code one simply reverses this procedure. For the Algorithms 1 and 2, we assume the data structure used to represent block sequences offers the methods *append*, *remove* and *indexOf*. $S.append(x)$ appends element x to sequence S , $S.remove(x)$ removes element x from sequence S , and $S.indexOf(x)$ returns the index of the first occurrence of element x in sequence S . While Algorithm 1 permutes a block sequence in order to get an encoding of message M , Algorithm 2 decodes a message from a given permutation by inverting the encoding technique used in Algorithm 1.

Theorem 3.1: The algorithm pair $(ENC_{order}, DEC_{order})$ transmits an optimal $\lfloor \log(|B|!) \rfloor$ bits over the request order channel, where B is the set of blocks requestable by the transmitting peer.

Proof: The transmission is correct since ENC_{order} implements a bijective function from the message domain $\{0, 1\}^{\lfloor \log(|B|!) \rfloor}$ to the domain of permutations of B . DEC_{order} implements the inverse bijection. The pair $(ENC_{order}, DEC_{order})$ is optimal because there are $|B|!$ many permutations of length $|B|$. Any algorithm pair can encode at most $\lfloor \log(|B|!) \rfloor$ bits in the order of the block sequence. ■

B. No Monitoring

A weakest authority is one that does not have the capacity to monitor connections at all. The only way such a limited authority can learn about illegal actions in the network is through reports of regular peers. As stated in the model section, we assume that if a conspirer u reveals its type to a regular peer v , then v will report u to the authority, and u will be punished. Thus, a conspirer u must not communicate using

plain text with a neighboring peer v unless it has verified that v is also a conspirer. If the verification is successful, u may send all subsequent messages to v in the clear. Hence, in order to solve $BROADCAST(M)$ it suffices to establish a connected graph among the conspirers where there is an edge between two conspirers $u, v \in C$ if they are neighbors and both of them know each other's type, i.e., both know that $u, v \in C$. Once connected, the message holder can send the message M in plain text to all of its neighboring conspirers, each of which will propagate it to its respective neighboring conspirers.

One straightforward approach to achieve this is to have the message holder $u \in C$ connect to every peer in P , determining every peer's type and then send the message M to the fellow conspirers, which are all directly connected to u . Although this simple approach solves $BROADCAST(M)$, the conspirers are well advised not to use it because of its lack of efficiency, both in terms of space and communication complexity. Since the message holder basically connects to the entire network it needs $\Omega(n)$ memory. Furthermore, this requires extensive polling of the public server that keeps track of all peers, and would cost $\Omega(n/k)$ messages as only k addresses are returned for each request. As k is typically a constant, this amounts to a communication complexity linear in n . Another major drawback of this scheme is that the server receives an exceedingly large number of requests, which basically gives away the identity of the message holder.

In contrast to this brute-force approach, we will now present a scheme that solves $BROADCAST(M)$ much more efficiently. The following lemma shows that, depending on the number of conspirers c , the number of connections that have to be established to ensure that all c conspirers induce a connected subnetwork is considerably lower than n .

Lemma 3.2: If each conspirer randomly acquires $8\frac{n}{c}\ln(nc)$ neighbors, then the subnetwork induced by the c conspirers is connected w.h.p.

Proof: First, we show that each conspirer u has a sufficiently large number of conspiring neighbors. Let $\mathcal{N}_u^c$ denote the set of neighbors of u that are conspirers. Since each neighbor v is chosen uniformly at random and the probability that $v \in C$ is c/n , we immediately have that $\mathbb{E}[|\mathcal{N}_u^c|] = 8\ln(nc)$. Using a standard Chernoff bound, we get that

$$\mathbb{P}[|\mathcal{N}_u^c| < 4\ln(nc)] = \mathbb{P}\left[|\mathcal{N}_u^c| < \frac{\mathbb{E}[|\mathcal{N}_u^c|]}{2}\right] \leq e^{-\frac{\mathbb{E}[|\mathcal{N}_u^c|]}{2^2 \cdot 2}} = \frac{1}{nc}.$$

Hence, the probability that *any* conspirer has fewer than $4\ln(nc)$ neighbors that are conspirers is upper bounded by $1/n$.

We now need to prove that this number of connections suffices to guarantee that all conspirers are connected with high probability. For this purpose, we use the following theorem about Erdős-Rényi random graphs $G(c, p_e)$ [1], [2]. $G(c, p_e)$ is a graph consisting of c nodes in which each of the $\binom{c}{2}$ edges is added to the graph independently with probability p_e .

Theorem 3.3 ([3]): If $p_e = (\ln c + t)/c$, then $G(c, p_e)$ is connected with probability $(1 + o(1))e^{-e^{-t}}$.

This theorem implies that if each edge is chosen with probability $\ln(nc)/c$, then the resulting graph is connected with probability at least $e^{-e^{-\ln n}} = e^{-1/n} \geq 1 - \frac{1}{n}$. If

Algorithm 3 BROADCAST(M)

```

1: repeat
2:   Add  $k$  random peers to neighbor set  $\mathcal{N}$ ;
3: until  $|\mathcal{N}| \geq 8\frac{n}{c}\ln(nc)$ 
4: Get  $6\log n$  random blocks in total from connected peers;
5:  $C := \emptyset$ ;
6: for each  $v \in \mathcal{N}$  in parallel do
7:   if REVEALTYPE( $v$ ) = conspirer then
8:      $C := C \cup \{v\}$ ;
9: if message holder then
10:  send  $M$  to all  $v \in C$ ;
11: else
12:  wait until message  $M$  received;
13:  send  $M$  to all  $v \in C$ ;

```

Subroutine REVEALTYPE(v)

```

14: wait until  $(|B_{u,v}| \geq 3\log n) \wedge (|B_{v,u}| \geq 3\log n)$ ;
15:  $B := \lceil 3\log n \rceil$  blocks of  $B_{u,v}$  with lowest indices;
16:  $B' := \lceil 3\log n \rceil$  blocks of  $B_{v,u}$  with lowest indices;
17: Sort  $B, B'$ ;
18:  $\Pi := ENC_{order}(B, \mathcal{H}(id_u || id_v || K, \lceil 3\log n \rceil))$ ;
19:  $\Pi' := ENC_{order}(B', \mathcal{H}(id_v || id_u || K, \lceil 3\log n \rceil))$ ;
20: for  $i := 0$  to  $|\Pi| - 1$  do
21:   for  $j := 0$  to  $|R_v| - 1$  do
22:     if  $R_{v,j} \neq \Pi'_j$  then return regular;
23:   if  $(|R_v| < i) \vee (|R_v| > i + 1)$  then
24:     return regular;
25:   else
26:     request  $\Pi_i$ ;
27:     wait until  $\Pi_i$  received;
28: return conspirer;

```

$p_e = \ln(nc)/c$, the expected number of neighbors of each node is $\ln(nc)$. Let L_u be the random variable that counts the number of u 's neighbors in a graph $G(c, p_e)$. Again using a Chernoff bound, it follows that

$$\mathbb{P}[L_u > 4\ln(nc)] = \mathbb{P}[L_u > (1+3)\mathbb{E}[L_u]] \leq e^{-\frac{3}{4}\mathbb{E}[L_u]} < \frac{1}{nc}.$$

The probability that *any* node has more than $4\ln(nc)$ neighbors in a graph of c nodes, where each edge is chosen with probability $\ln(nc)/c$, is upper bounded by $1/n$. This means that we also get a connected graph, with high probability, if each node chooses $4\ln(nc)$ neighbors uniformly at random in a graph of size c . We already established that by connecting to $8\frac{n}{c}\ln(nc)$ random neighbors, each conspirer connects to at least $4\ln(nc)$ conspirers with high probability, i.e., each conspirer implicitly chooses at least $4\ln(nc)$ random neighbors in the conspirer subgraph. Therefore, the subnetwork is connected w.h.p. ■

Lemma 3.2 states that if $c \in \Theta(n)$, acquiring a logarithmic number of neighbors is sufficient for the conspirers to end up in a connected component. Note that the constant 8 can probably be reduced using more elaborate arguments. However, it is clear that the asymptotic behavior is correct for any $c \in \Theta(n)$ as the graph $G(n, p_e)$ is *not* connected if $p_e = ((1 - \epsilon)\ln n)/n$ for any $\epsilon > 0$ asymptotically almost surely [2].⁸

Putting all the building blocks together, we are able to give an algorithm that solves $BROADCAST(M)$, which is executed at each conspirer $u \in C$. Algorithm 3 first polls the public server until it has enough neighbors to ensure that all

⁸“Asymptotically almost surely” means that the probability that the claimed bound holds tends to 1 as $n \rightarrow \infty$.

conspirers are in a connected component. In a second phase, each conspirer u gathers enough blocks to make sure that any two conspirers have enough trading blocks to determine each other's type. If another peer connects to u in this phase, u reports that it does not have any blocks yet in order to avoid trading blocks with other conspirers prematurely. Subsequently, each conspirer starts requesting blocks from all its neighbors and thereby revealing its type by transmitting a secret key over the request order channel. Since the number of required blocks is relatively small, i.e., we assume that $6 \log n \ll m$, each peer can accumulate this number of blocks quickly. Once the message holder knows all its neighbors' types, it sends message M in plain text to its neighboring conspirers. All other conspirers wait for M and pass it to their neighboring conspirers as soon as they receive it.

For the subroutine REVEALTYPE we assume that another thread run by u listens to neighbor v , and whenever it receives a block request b_x from v , appends b_x to a list R_v , and starts transmitting block b_x to v . Moreover, we denote by $B_{u,v}$ the set of blocks that u can request from v , i.e., the set of blocks that v claims to possess and u does not. The following theorem states the communication and the space complexity of this broadcast algorithm.

Theorem 3.4: If $c \in [18 \ln n, n/3]$ and $m \geq (24e+6) \log n$, Algorithm 3 secretly broadcasts a message M of arbitrary length in an unmonitored network w.h.p. The space complexity is $\mathcal{O}(\frac{n}{c} \log n + |M|)$ and the communication complexity is $\mathcal{O}(\frac{n}{c} \log n + \log^2 n + |M| \log n)$ w.h.p.

Proof: We start by showing that each conspirer has to collect only $12(1+o(1))\frac{n}{c} \ln(nc) \in \mathcal{O}(\frac{n}{c} \log n)$ peer addresses in order to get $8\frac{n}{c} \ln(nc)$ distinct random neighbors, i.e., it has to inquire the server only $\mathcal{O}(\frac{n}{c} \log n)$ times. Let the random variable N_i indicate the number of random peers that have to be collected to get the i^{th} distinct neighbor after having collected $i-1$ distinct neighbors. It holds that $\mathbb{E}[N_i] = \frac{n}{n-i+1}$. Let $T_j = \sum_{i=1}^j N_i$ be the random variable that indicates how many random neighbors have to be collected until j distinct peers have been discovered.

Since $c \geq 18 \ln n > 9 \ln(nc)$, and given that we need $j = 8\frac{n}{c} \ln(nc)$ distinct neighbors, it holds that $n-j > n/9 \in \Omega(n)$. Hence, we have that

$$\begin{aligned} \mathbb{E}[T_j] &= \sum_{i=1}^j \mathbb{E}[N_i] \\ &= n \left(\frac{1}{n} + \frac{1}{n-1} + \dots + \frac{1}{n-j+1} \right) \\ &= n(H_n - H_{n-j}) \\ &\leq n(\ln(n) - \ln(n-j)) \\ &= -n \ln \left(1 - \frac{j}{n} \right) = -\ln \left(1 - \frac{j}{n} \right)^n \\ &\leq -\ln(e^{-j}) + o(1) = j + o(1), \end{aligned}$$

where H_n is the n^{th} harmonic number. We can again use a Chernoff bound to see that

$$\mathbb{P}[T_j > (1 + 1/2)\mathbb{E}[T_j]] < e^{-\frac{1}{16}\mathbb{E}[T_j]} \leq e^{-3/2 \ln(nc)} < \frac{1}{nc}.$$

Thus, each conspirer has to acquire fewer than $12 \cdot (1 + o(1))\frac{n}{c} \ln(nc)$ random peer addresses with high probability. Since each node further has to store the message M , the bound on the space complexity follows.

Next, we argue that a conspirer can acquire $6 \log n$ blocks quickly (cf. Line 4). Recall that $\mathcal{N}_u^c$ denotes the neighboring conspirers of u , and that $\mathbb{E}[|\mathcal{N}_u^c|] = 8 \ln(nc)$. By means of a Chernoff bound we can see that

$$\mathbb{P}[|\mathcal{N}_u^c| > 2\mathbb{E}[|\mathcal{N}_u^c|] \leq e^{-\frac{\mathbb{E}[|\mathcal{N}_u^c|]}{4}} = \frac{1}{(nc)^2},$$

i.e., any conspirer has fewer than $16 \ln(nc)$ conspiring neighbors and hence more than $8\frac{n}{c} \ln(nc) - 16 \ln(nc) > 6 \log n$ regular neighbors with high probability. This implies that fewer than one block has to be acquired on average per regular neighbor, which can be accomplished swiftly.

We proceed by proving that $6 \log n$ blocks suffice to ensure that each conspirer has $3 \log n$ blocks to trade with each neighboring conspirer with high probability, which is the number of requestable blocks required for the subroutine REVEALTYPE to work (cf. Line 14–16). Note that we can transmit $\log((3 \log n)!) \geq 3 \log n$ bits over the request order channel by using $3 \log n$ blocks. As argued before, exchanging $3 \log n$ bits with a conspirer is enough to safely verify its type. Thus, we have to show that the probability that two conspirers have more than $3 \log n$ blocks in common is negligible. The probability that the i^{th} block is also in the set of blocks that another conspirer acquires is at most $(6 \log n)/(m-i) \leq (6 \log n)/(m-6 \log n)$ because at most all $i-1$ previous blocks are not in the other conspirer's set. Let the random variable X denote the number of such colliding blocks and let $p := (6 \log n)/(m-6 \log n)$. We can upper bound the probability that $X \geq 3 \log n$ as follows. Let $\mathcal{X}_{u,v}$ be the set of colliding blocks, i.e., the blocks acquired by both u and v .

$$\begin{aligned} \mathbb{P}[X \geq 3 \log n] &= \sum_{i=3 \log n}^{6 \log n} \mathbb{P}[|\mathcal{X}_{u,v}| = i] \\ &\leq \sum_{i=3 \log n}^{6 \log n} \binom{6 \log n}{i} p^i (1-p)^{6 \log n - i} \\ &\leq \left(\frac{6 \log n}{3 \log n} \right) p^{3 \log n} \\ &\leq (2e)^{3 \log n} p^{3 \log n} \\ &= \left(\frac{12e \log n}{m-6 \log n} \right)^{3 \log n}. \end{aligned}$$

Since $m \geq (24e+6) \log n$, we have that $\mathbb{P}[X \geq 3 \log n] \leq 1/n^3$ and thus every conspirer has enough blocks to request from all other conspirers with high probability. This proves that each conspirer can identify the conspirers among its neighbors.

The question remains how many messages a conspirer u has to send in the REVEALTYPE phase. As u requests one block and then waits for the next request of the other party, u can identify regular peers quickly: u identifies a regular peer v as soon as v requests the “wrong” block. Since the probability that v requests the correct first block is $1/(6 \log n)$ and the probability that each subsequent block

is also “guessed” correctly is smaller than $1/2$ (until only two blocks are left), the expected number of blocks that need to be exchanged is lower than 2. In total, the number of messages exchanged with regular peers is thus $\mathcal{O}((n/c) \log n)$. It is easy to see that this bound also holds with high probability (again using a standard Chernoff-type argument). As for any conspirer u , $|\mathcal{N}_u^c| \leq 16 \ln(nc)$ with high probability, all conspirers exchange at most $3 \log n \cdot 16 \ln(nc) \in \mathcal{O}(\log^2 n)$ blocks with other conspirers with high probability. Each conspirer must further send the message M to the other conspirers in its neighborhood, which costs at most $|M| \cdot 16 \ln(nc) \in \mathcal{O}(|M| \log n)$ messages. If we combine the bound on the number of messages required to gather enough neighbors, identify the conspirers, and broadcast M , we get the claimed bound on the message complexity. ■

Note that for a small c , i.e., $c \in \mathcal{O}(\log n)$, each conspirer has to connect to $\Omega(n)$ random peers to establish a connected conspirer component. In particular, if $c < 8 \ln n$, each conspirer connects to all n peers for large n with Algorithm 3. In this case, the conspirers can resort to the aforementioned brute-force approach, especially when considering that in *any* broadcast algorithm, a peer must connect to at least $\Omega(n/c)$ peers to find another conspirer. Note also that if the conspirers only have an estimate of n and c , they can increase the number of neighbors and the length of the exchanged key (continuously) by an appropriate factor to ensure that the presented algorithms still succeed w.h.p.

C. Individual Monitoring

Let us now consider an authority that is able to monitor connections individually. By *individually* we mean in this context that the authority can monitor any communication link between any two peers; however, it is not capable of correlating the data gathered at different connections. As we will see, the adversary in this model is stronger in that the size of the message M that can be transmitted depends on the total number of blocks m . In other respects this model is quite similar to the setting in the previous section where the authority acted only as a punitive deterrent. In particular, as long as a conspirer u does not know a neighboring peer’s type, it does not make a difference whether or not the link to that peer is monitored as the hidden channel must be used to communicate.

The reason why the size of the message M is limited is that after a conspirer has successfully revealed another conspirer’s type, it cannot communicate freely over this link since the monitoring authority could detect this illegal communication. Hence, hidden communication must also be used to transfer the message M . Furthermore, a conspirer v must not request the same file block twice from the same neighbor since the monitoring authority would realize that v is requesting a block it has already received. We can conclude that the maximum size of the message M depends on the number m of blocks as there is no need for additional communication between any two peers once m blocks have been sent in both directions. Consequently, this setting forces the conspirers to use as little blocks as possible for each individual communication link. On a particular link (u, v) , however, conspirer u can still

underreport on the blocks that it has received from peers other than u , and it can re-request blocks that it has already received from other peers.

In the following, we will outline how to adapt Algorithm 3 for this setting. Since each conspirer can still determine its neighbors’ types using the REVEALTYPE mechanism, the first part of the algorithm (more precisely, Line 1–8) remain unchanged. Instead of immediately sending message M in plain text, a conspirer u downloads more blocks from its regular neighbors until it has $\Theta(m)$ blocks, preferably all m blocks. In the next step, it reports to each connected conspirer v half of the file blocks that it has gathered in the previous phase. In order to maximize the number of tradable blocks, u and v should report (mostly) disjoint block sets. This can easily be accomplished by hashing the node identifiers and the secret key to get a sequence of blocks that, e.g., the node with the smaller identifier offers. The other peer performs the same computation and simply offers the blocks not in the sequence. If the two sets are completely disjoint, this may also raise suspicion. Therefore, it may make sense to enlarge the set of offered blocks. This way, each conspirer can exchange $\Theta(m)$ blocks with its conspiring neighbors. Note that for this method to work, each conspirer must remember which blocks it has traded with each of its neighbors in the REVEALTYPE phase in order to avoid re-requesting blocks on individual links. We call this adaptation of the broadcast method $\mathcal{ALG}_{individual}$, for which we can show the following.

Theorem 3.5: If $c \in [18 \ln n, n/3]$ and $m \geq (24e+6) \log n$, $\mathcal{ALG}_{individual}$ secretly broadcasts a message of $\Theta(m \log m)$ bits in an individually monitored network w.h.p.

Proof: Any conspirer u is able to use $(m - |\mathcal{X}_{u,v}| - R)/2$ blocks for transmitting message M to neighbor v , where $\mathcal{X}_{u,v}$ is the set of colliding blocks between u and v in the REVEALTYPE phase, and R is the number of blocks used by REVEALTYPE. From the proof of Theorem 3.4 we know that $|\mathcal{X}_{u,v}| + R$ is at most $6 \log n$ with high probability. Hence, conspirer u can use at least $(m/2) - 3 \log n \in \Theta(m)$ blocks to transmit $\Theta(\log(m!)) = \Theta(m \log m)$ bits to neighbor v over the request order channel. ■

D. Complete Monitoring

If the *complete* network is monitored, i.e., the authority monitors all communication links and may also correlate data gathered at different links, it gets considerably harder to exchange secret messages. The main difference to individually monitored communication is that a conspirer can no longer underreport, or request a block that it has already received from another peer. In general, the more blocks a peer possesses, the more constrained it is in its actions. Consequently, we have to impose tighter restrictions on the number of conspirers and on the number of blocks in order to enable the conspirers to receive the secret message. Note that the specific restrictions we impose are used for ease of presentation and stronger bounds are again possible by means of a more complicated analysis. Given these conditions, we can still transmit a fairly long message without modifying $\mathcal{ALG}_{individual}$ substantially: The only modification to Algorithm 3 is that every conspirer acquires $8\sqrt{n} \ln(nc)$ random blocks instead of only $6 \log n$.

We call this adaptation of the algorithm $\mathcal{ALG}_{complete}$, for which we get the following result.

Theorem 3.6: If $c \geq \sqrt{n} \geq 6$ and $2060\sqrt{n} \ln^2(nc) \leq m \in \Theta(n)$, algorithm $\mathcal{ALG}_{complete}$ secretly broadcasts a message of $\Theta(\sqrt{m} \log^2 m)$ bits in a completely monitored network w.h.p.

Due to lack of space, the proof of Theorem 3.6 is omitted and can be found in the accompanying technical report.⁹

E. Stochastic Monitoring

In the previous sections, we assumed that permuting the request sequence order does not arouse suspicion. In reality, there may be certain policies that restrict such behavior, e.g., it may be common practice to request the least frequently advertised block first (rarest-first) in order to keep all blocks available as long as possible. Streaming applications may demand even stricter policies; the most extreme restriction would be to acquire all blocks in ascending order, which would prohibit using the request order channel completely. However, as requesting blocks in ascending order is not an efficient dissemination scheme, peers typically have the freedom to (randomly) request any block in a certain window. Moreover, if the rarest-first policy is used, the decision which block to request next is made locally, which means that it is not easily possible for a peer to verify that a block that a neighbor has requested is indeed the rarest according to this neighbor's local view. Nonetheless, there may be certain request patterns that raise suspicion. If the authority is aware of all legal strategies, it has additional power to expose conspirers. In reference to related work in the field of steganography, e.g. [4], we assume in this section that regular peers choose their request order permutation according to a distribution $\mathcal{C}$, and that all peers as well as the authority know $\mathcal{C}$.¹⁰ As a consequence, a monitoring authority can assign to each request order permutation Π a probability $p(\Pi)$ that Π was generated according to $\mathcal{C}$. If the permutations by a peer u deviate with statistical significance from $\mathcal{C}$, i.e., if $p(\Pi)$ is below a certain threshold ϵ , then the authority might classify u as a conspirer. Choosing a reasonable ϵ , however, is a delicate task. A careful authority may want to prevent false positives in any case, and thus choose $\epsilon = 0$. This implies that the conspirers have to avoid all permutations with probability mass 0 and, if there are such permutations, the capacity of the request order channel is reduced. If a non-zero ϵ is chosen and there are permutations Π with $0 < p(\Pi) < \epsilon$, the authority reduces the request order channel's capacity even more. On the other hand, the authority risks punishing regular peers by increasing ϵ . One approach for the conspirers to adapt to such stochastic monitoring is to come up with an adapted mapping of messages to the set of valid permutations, i.e., permutations Π with $p(\Pi) > \epsilon$, or, as this is rather cumbersome, they might use a generic approach such as Algorithm 4. Algorithm 4 repeatedly XORs the original message M with a bitstring produced by a pseudo-random generator $\mathcal{K}$ and maps this string to a permutation until

Algorithm 4 $ENC_{stochastic}$

```

1:  $i := 0$ ;
2: repeat
3:    $\Pi := ENC_{order}(M \oplus \mathcal{K}(i)||i)$ ;
4:    $i++$ ;
5: until  $p(\Pi) > \epsilon$ 
6: return  $\Pi$ ;
```

a valid permutation is generated. Note that $\mathcal{K}$ is deterministic, and therefore a receiving conspirer can revert $ENC_{stochastic}$ by applying DEC_{order} , extracting i , and XORing the result with $\mathcal{K}(i)$. In order to extract i , the conspirers have to either fix the number of bits used for i or introduce a preamble marking its beginning. The running time of $ENC_{stochastic}$ depends on the distribution $\mathcal{C}$ and the threshold ϵ , and is in the order of the ratio of invalid to valid permutations. The advantage of such a generic approach is that it can even be applied in a completely monitored network, where the validity of a request order permutation generally depends on the requests already sent to other neighbors.

For the special case of $\mathcal{C}$ being the uniform distribution, a standard OAEP scheme [5], [6] would also be sufficient to make the permutations look unsuspicious. As OAEP includes random bits, it would furthermore have the property that when a message is sent over the same link several times the produced request permutation will always look different (unless the same random bits accidentally occur multiple times).

F. Additional Steganographic Channels

All schemes presented so far make use of the request order channel. This section discusses additional steganographic channels that could be exploited for hidden communication.

Subset Selection. Algorithm 3 hides information by requesting a permutation of the $s := 3 \log n$ requestable blocks with lowest index. A conspirer u can transmit an additional unused $\lfloor \log \binom{B_u}{s} \rfloor$ bits to v by selecting the subset of s blocks to be requested according to a shared secret.

Timing. The protocol allows to introduce some variation in the timing of the protocol messages, as peers are not expected to request blocks or to answer requests immediately. Hence, conspirers can hide information by delaying protocol messages. One possibility is, e.g., to encode information in the time between the reception of a block request and the corresponding transmission. Note that such steganographic channels are only feasible if the connection between the peers is stable, i.e. the message delays are within a reasonable range. In realistic networks, conspirers would most likely have to use error-correcting codes. Generally, the capacity of such time-coded channels depends on the accuracy of the measurements, the predictability of delays, and on the extent to which a conspirer may delay the protocol without evoking suspicion.

Bandwidth. A conspirer might vary the rate at which it sends file blocks, or network packets in general. One possible protocol would be to encode bits in the transitions from one rate to another. With each transmission, a peer either goes from a low to a high bitrate to send a 1, or from a high to a low bitrate to send a 0. Note again that in practice one could use error-correcting codes to account for unstable connections.

⁹See www.dcg.ethz.ch/publications/infocom2011ELWtik.pdf.

¹⁰As noted in [4], the assumption that such a distribution is known or that there is at least an oracle available that generates permutations according to $\mathcal{C}$ is often critical. In a p2p context, however, it is reasonable to a certain extent as there is only a finite number of clients that implement a certain protocol.

Ports. Another channel is the choice of the communication port. Unfortunately, this channel is not scalable since the port for the communication is only chosen once per connection, and its capacity is rather small. More importantly, many peers are typically not able to use this channel since they are behind a NAT router that allows no explicit control of the ports.

Encryption. Recently, many p2p networks introduced the possibility to encrypt the messages between the peers by means of public key, private key encryption. The open source bittorrent client Vuze [7], e.g., implements RC4 [8] encryption, primarily to avoid aggressive traffic shaping by internet service providers that prefer client-server traffic to p2p traffic. Encryption is a straightforward technique to hide communication: Once two conspirers are connected, they can use encrypted messages to communicate. However, if the authority has the power to monitor individual links, it could still detect illegal communication if more than m blocks are sent in either direction. Thus, encryption does not help in this setting in the sense that also at most $\Theta(m \log m)$ bits can be broadcast.

IV. IMPLEMENTATION

Some techniques described in the previous section have been implemented and are now used in the context of file sharing. In particular, we incorporated a steganographic handshake in our own, publicly available BitTorrent client, BitThief [9].

The BitThief client emerged as a proof of concept to show that free riding in BitTorrent is possible [10], i.e., BitThief downloads without uploading. As the goal of the BitThief project is now to improve BitTorrent’s incentive compatibility, we designed an enhanced tit-for-tat (T4T) protocol [11] that can be used among instances of the BitThief client to exchange files once they have found each other. A high download rate is achieved by using the T4T protocol for BitThief instances and the regular BitTorrent protocol when communicating with other clients. In order not to be identified as a BitThief, and thus risk being banned by other clients, BitThief must reveal its identity only to fellow BitThief instances, i.e., it has to solve the REVEALTYPE problem. BitThief solves REVEALTYPE by impersonating standard BitTorrent clients and performing a steganographic handshake.

Unfortunately, we could not solve REVEALTYPE exactly as discussed in Section III-A, since our practical solution has to satisfy the additional constraint that BitThief never uploads a file block to a client that is not another BitThief instance. Note that a free riding client must not announce having file blocks that a connected peer v is interested in as v may request such a block. Upon receiving a valid request from v , a free rider would have to upload data, and violate the no-upload constraint, or ignore the request and risk revealing its identity. Moreover, many clients wait until they receive a requested block before they send out more blocks, i.e., a free rider would not receive any blocks from this peer anymore for free until the request is served. Consequently, when two potential BitThief clients meet, they both announce that they have no file blocks yet, and none of them can make a block request.

Our solution to REVEALTYPE in this setting is to abuse an extension of the BitTorrent protocol called “peer exchange” (PEX). This extension allows peers to learn about other peers

from known peers without inquiring a tracker. Basically, a PEX message contains a list of some of the peers to which the sender is connected, and optionally also a list of dropped peers, i.e., peers that cannot be reached (anymore). We use the order of this peer list as a steganographic channel. In particular, if an instance u of BitThief wants to determine whether another peer v is also an instance of our client, it sends a PEX message where the peer list is permuted with respect to its natural order according to a hash $\mathcal{H}(id_u || id_v || F || K)$ of u and v ’s identifiers¹¹, hashed meta data F of the file that is exchanged, and a secret K that is shared by all BitThief clients. The permutation is constructed along the lines of Algorithm 1. Upon receiving a PEX message from v , u checks whether the peer list is ordered according to $\mathcal{H}(id_v || id_u || F || K)$. If the check is successful v sends back a PEX message to u with a list ordered according to $\mathcal{H}(id_u || id_v || F || K)$, unless it has already sent a PEX message with a hidden key to v before. After successfully checking a received PEX message for the hidden key, BitThief switches to the T4T protocol, and starts trading with the connected BitThief instance. As with the request order in Section III, the PEX protocol does not impose a specific peer list order, and other clients do not interpret it in any way. Hence, a BitThief client does not evoke suspicion when sending such a modified PEX message to other clients.

Note that there may be a problem if the swarm is small and consequently the length of the list in a PEX message is short. In this case, the probability that another client sends a list ordered correctly according to the hashing algorithm “by accident” is high. Therefore, our client resorts to a fallback strategy when it knows fewer than 17 peers. It creates a bogus entry in the list (of active neighbors) by hashing F , K , the recipient’s and its own identifier, and then constructing an IP address and a port out of this hash. The recipient can compute the same fake entry and easily check whether this entry is in the received list. Since there are roughly 2^{48} possible fake addresses consisting of a (32-bit) IP address and a (16-bit) port,¹² the probability of false positives is sufficiently small. We chose 17 as the threshold for the fallback strategy because $\log(17!) = 48.34 > 48$, and thus permuting a list of at least 17 entries can hide more information than one fake IP address and port. Note that it is possible that in future versions of PEX the order of entries will be predetermined. While permutations could no longer be used to establish a hidden handshake, one could still add bogus clients to the list to solve this problem. It is unlikely that a regular peer identifies a faked entry in the PEX message, even if it realizes that the corresponding peer is not available, as it is plausible that some peer may be connected to a peer u , but another peer cannot contact u , e.g., because it is behind a NAT router and port-forwarding is not enabled, or because this peer left the swarm in the meantime. Thus, if a peer does not get a response from some peers in the list, it cannot conclude that the sender of the list sent invalid information. As a tracker theoretically knows every peer that has entered the swarm, also this vulnerability could be closed by having the tracker verify

¹¹In BitTorrent 20-bit peer identifiers are used.

¹²Some ranges of IP addresses and some ports cannot be used, which slightly reduces the address space.

each PEX message. However, in practice, a tracker is often not aware of all peers in the swarm because a torrent typically offers several trackers as entry points. More importantly, asking the tracker to verify the integrity of PEX messages defeats the purpose of these messages to reduce the load of the tracker.

Extensive tests in live BitTorrent swarms have shown that the hybrid steganographic handshake works well, i.e., the BitThief clients successfully find each other and switch to their private T4T protocol without being detected.

V. RELATED WORK

The study of steganography in computing dates back to 1983 when [12] formulated the prisoner's problem where two prisoners must hatch an escape plan while being monitored by the warden. Hopper studies steganography from a cryptographic perspective in [4]. Chapter 3 on symmetric-key steganography investigates the setting where two parties that share a secret would like to exchange hidden messages over a monitored public channel. This setting differs from our work in that both parties are mutually aware of each other's type, and that the communication is not limited. Cachin [13] proposes an information-theoretic model, and provides a universal information hiding scheme for the symmetric-key setting. Contrary to our work, public-key steganography [4], [14] studies the setting where the sender and receiver do not share a secret key. Another branch of steganography studies information hiding in media files such as images or movies. Media files are especially suited for hiding information since they are large and since human perception easily fails to detect minor modifications (see [15] for a high level overview). Steganography is also used in digital rights management (DRM) and digital watermarking.

Closely related research on *secret handshakes* was conducted by [16] and followed, among others, by [17], [18]. Secret handshakes are protocols that allow the participants to establish a secure and anonymous communication channel only if they are all members of the same group. In contrast to the settings studied in [16]–[18], where the secret handshake is interwoven with an ordinary handshake, a conspirer in our setting cannot initiate a secret handshake by tweaking the ordinary handshake, or by sending an additional message over the standard communication channel, because this would be an illegal, observable deviation from the imposed protocol. We overcome this problem by means of steganographic channels. In that sense, our steganographic handshake can be viewed as a secret handshake conducted on steganographic channels.

Research on anonymity in networks relates to our work in that anonymity also facilitates committing illegal actions without being punished by an authority. However, whereas anonymity and privacy have been classic design goals of many p2p systems (e.g., Freenet [19]), there has been little research on steganography in the context of p2p networks. Most existing work essentially uses steganography to hide information in (large) files. In particular, [20] proposes to use watermarking for copyright protection in p2p systems, and [21] proposes to hide information in torrent meta files. Arguably the most related work is due to Bickson [22] who showed how to hide content in Gnutella queries to broadcast a message that can only be decrypted by peers that share the sender's secret. In

contrast to his work, we do not hide information in media or torrent files, nor in query messages, but exploit the protocol itself. Furthermore, our approach also works in (BitTorrent-like) overlay networks where no lookup mechanism exists. Finally, we also study the feasibility of steganographic handshakes and broadcasts under several monitoring levels.

VI. CONCLUSION

We disclosed several steganographic channels in p2p protocols and successfully exploited them to achieve a steganographic handshake and broadcast in BitTorrent-like p2p systems. Most of the discussed channels can be encountered not only in p2p protocols, but in many network protocols in general. Permuting packet sequences and introducing artificial delays are potential mechanisms for hidden communication in any network protocol. Our techniques can be naturally adapted or extended to secretly communicate in a variety of network protocols. The assumption that a shared secret or, for that matter, a publicly available software client can be kept closed-source is somewhat problematic. Hence, an interesting question is whether and how one can get rid of this assumption.

REFERENCES

- [1] P. Erdős and A. Rényi, "On Random Graphs," *Publicationes Mathematicae*, vol. 6, pp. 290–297, 1959.
- [2] —, "On the Evolution of Random Graphs," *Publ. Math. Inst. Hungar. Acad. Sci.*, vol. 5, pp. 17–61, 1960.
- [3] R. van der Hofstad, "Random Graphs and Complex Networks," *Unpublished manuscript*, 2007.
- [4] N. J. Hopper, "Toward a Theory of Steganography," Ph.D. dissertation, Carnegie Mellon University, Pittsburgh, PA, USA, 2004.
- [5] M. Bellare and P. Rogaway, "Optimal Asymmetric Encryption," in *Advances in Cryptology — EUROCRYPT'94*, 1995.
- [6] V. Boyko, "On the Security Properties of OAEP as an All-or-Nothing Transform," in *Advances in Cryptology — CRYPTO'99*, 1999.
- [7] <http://azureus.sourceforge.net/>.
- [8] R. Rivest, "The RC4 Encryption Algorithm," *RSA Data Sec. Inc.*, 1992.
- [9] <http://bitthief.ethz.ch/>.
- [10] T. Locher, P. Moor, S. Schmid, and R. Wattenhofer, "Free Riding in BitTorrent is Cheap," in *Proc. 5th Workshop on Hot Topics in Networks (HotNets)*, Irvine, California, USA, November 2006.
- [11] T. Locher, S. Schmid, and R. Wattenhofer, "Rescuing Tit-for-Tat with Source Coding," in *Proc. 7th IEEE International Conference on Peer-to-Peer Computing (P2P)*, September 2007.
- [12] G. J. Simmons, "The Prisoners' Problem and the Subliminal Channel," in *Advances in Cryptology — CRYPTO'83*, 1984.
- [13] C. Cachin, "An Information-theoretic Model for Steganography," *Information and Computation*, vol. 192, no. 1, pp. 41–56, 2004.
- [14] M. Backes and C. Cachin, "Public-Key Steganography with Active Attacks," in *Proc. 2nd Theory of Cryptography Conference (TCC)*, 2005.
- [15] G. C. Kessler, "An Overview of Steganography for the Computer Forensics Examiner," vol. 6, no. 3, 2004.
- [16] D. Balfanz, G. Durfee, N. Shankar, D. Smetters, J. Staddon, and H.-C. Wong, "Secret Handshakes from Pairing-Based Key Agreements," in *Proc. IEEE Symposium on Security and Privacy (SP)*, 2003.
- [17] G. Tsudik and S. Xu, "A Flexible Framework for Secret Handshakes," in *Proc. 6th Workshop on Privacy Enhancing Technologies (PET)*, 2006.
- [18] S. Jarecki and X. Liu, "Private Mutual Authentication and Conditional Oblivious Transfer," in *Advances in Cryptology — CRYPTO'09*, 2009.
- [19] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong, "Freenet: A Distributed Anonymous Information Storage and Retrieval System," in *Int. Workshop on Designing Privacy Enhancing Technologies*, 2001.
- [20] D. Tsolis, S. Sioutas, and T. Papatheodorou, "Digital Watermarking in Peer to Peer Networks," in *Proc. 16th International Conference on Digital Signal Processing (DSP)*, 2009.
- [21] Z. Li, X. Sun, B. Wang, and X. Wang, "A Steganography Scheme in P2P Network," in *Proc. 4th International Conference on Intelligent Information Hiding and Multimedia Signal Processing (IIH-MSP)*, 2008.
- [22] D. Bickson, "Steganographic Communications Using the Gnutella Network," Master's thesis, Hebrew University of Jerusalem, 2003.