No Junk, no Peeking, Serious Offers Only: P2P File Exchange in Wireless Ad-hoc Networks

Ryan Vogt
Computing Science Dept.
University of Alberta
Edmonton, Alberta
CANADA T6G 2E8
e-mail: vogt@cs.ualberta.ca

Ioanis Nikolaidis
Computing Science Dept.
University of Alberta
Edmonton, Alberta
CANADA T6G 2E8
e-mail: yannis@cs.ualberta.ca

Pawel Gburzynski
Computing Science Dept.
University of Alberta
Edmonton, Alberta
CANADA T6G 2E8
e-mail: pawel@cs.ualberta.ca

Abstract

We consider the problem of anonymous file exchange in wireless ad-hoc networks. We propose a new protocol, dubbed RASH (Roam And SHare), to address the problem. RASH is a peer-to-peer (P2P) protocol which fulfills three objectives: (a) protection of user privacy (understood as the confidentiality of their interests), (b) support for a high degree of file authentication and integrity (to reduce or eliminate false downloads and spoofs), and (c) enforcement of reciprocity (to remove freeloaders from the system). The environment of our scheme is purely ad-hoc, in that it does not assume pre-established trust, fixed-infrastructure, or the existence of specialized nodes.

1. Introduction

In a pure mobile ad-hoc network environment, centralized trust authorities are not convenient, since both node connectivity and the user population are dynamic and unknown a priori. We consider the problem of anonymous file exchanges between users in this environment. Files are known to users by attributes, which for simplicity, we will call *file names*. Our primary objective is to preserve as much peer anonymity as possible. In particular, a node *B* that does not have a copy of the file *F* being sought by another node *A* is not able to tell that *A* is seeking *F* if it overhears, or even assists in, the transaction. On top of this, we would like to offer *A* some confidence in what is being downloaded. This means that before *A* decides to receive *F* from some node *C*, it would like to make sure that *C* is in fact in possession of *F*.

In a wireless environment, as opposed to a wired P2P system, it is particularly crucial to avoid false downloads, which waste precious battery power. While absolute certainty that *C* possesses *F* may be difficult to attain, a node will be able to increase its certainty to an arbitrary degree by (anonymously) querying its neighbourhood. We would also like to stimulate a communal behaviour, i.e., make sure that any downloading node has a strong incentive to offer its files to other nodes, and that it is no less strongly discouraged from playing pranks or cheating. Moreover, due to node mobility, data exchange episodes are bound to be short and unreliable. Generally, one cannot hope to receive a complete file (which can be large) in a single session. Consequently, we assume that files are naturally split into *segments*, and that a single download episode deals with the acquisition of one segment. It is possible to acquire the requisite segments of a file in any order (and at any time) and then reassemble the complete file at the end.

An Internet–based (wired) P2P file exchange system can afford complex techniques for preserving the anonymity of peers. For example, it can resort to mix networks [3, 4, 8] to cryptographically guarantee the untraceability of the message source and disguise the message from all intermediate nodes. All solutions in this class assume three prerequisites: stability of the network infrastructure, pre-established trust of at least some participants, and bandwidth redundancy, which are all absent in an ad-hoc system. Even on the Internet, because of the bandwidth requirements, this kind of anonymity enforcement is not a popular solution for file transfers, and is typically confined to low-bandwidth applications, e.g., electronic mail. Notably, anonymity in popular wired P2P systems like Napster and Kazaa is achieved by obfuscation and volume rather than cryptography.

One important design decision of our protocol is to restrict the P2P data transfers to take place between *directly* communicating peers, i.e., single-hop neighbours in the ad-hoc network. That is, we build the protocol relying solely on the interaction of nodes within a common wireless Local Area Network (LAN). Only the acquisition of file information, as opposed to the actual data transfers, may span more than one–hop (though even this allowance is not a requirement). One reason for restricting communication to immediate neighbours is that actions of a file recipient are inherently kept more private when communicated exclusively to peers within a limited physical range. At least this is the case once we agree that the truly complex solutions, like mix networks, are not feasible in the wireless environment. Quite often, users within the single-hop radius are also able to maintain visual contact, which can be reassuring (it plays the role of "soft authentication" in traditionally understood personal connections). On a more practical note, owing to the limited bandwidth and battery power, we should strive to keep the number of packets transmitted within the network to a minimum. Routing the file exchange traffic, which may deal with fat files transmitted in the background of other applications, would severely oversubscribe our resources, especially if the forwarding nodes, by not knowing the contents of what they forward, were unable to benefit from their service.

It seems to us that the whole concept of P2P file exchange in an ad-hoc wireless network only makes sense when viewed as cooperation between peers within their direct transmission range. A user willing to receive a certain (possibly long) file can be expected to put up with the energy expense required to accomplish this transaction. Similarly, the file-provider, by the fact of possessing the file and thus implicitly submitting to the social rules of reciprocity, can be expected to absorb the cost of delivery. But spreading the transfer over a possibly large population of intermediate relays would likely result in committing most of the network bandwidth and energy to this particular application, which some (uninterested) nodes might rightfully perceive as frivolous. One can envision a wireless P2P file exchange system as a social hobby: people with similar interests are likely to frequent the same places, which will couple their physical proximity to the likelihood of having something to share. The nomadic premise of wireless networking will correspond with the nomadic habits of people sharing their interests and hobbies.

In the next section, we outline the basic ingredients used in the design of the RASH (Roam And SHare) protocol. Section 3 elaborates on how the information regarding file segments is acquired anonymously and outlines two strategies with different levels of appropriateness for wireless environments. Section 4 details the means by which the transfer (or exchange) of file segments is to take place. Our objective for high throughput, low delay transfer leads us to different schemes from those proposed for file information acquisition. In Section 5 we describe a variety of enhancements in the form of trust models that are suitable for RASH. We conclude in Section 6 with a description of certain open issues.

2. Outline

Following the tradition of presenting cryptographic protocols, let us introduce Alice. Alice builds her collection of files from separate segments, which are generally acquired from different peers. We would like to attain the following property: Carol, who does not possess the requested file, not only is unable to send junk that she pretends is valid to Alice; she cannot even find out what Alice is asking for, or determine if and when the particular segment requested by Alice is being transmitted. This set of postulates can be succinctly expressed as "no junk, no peeking, serious offers only." Clearly, in order to ensure the seriousness of a node's (Bob's) offer, Alice must acquire (from elsewhere) more information about the requested file than its name-like description. She has to know something about the file and its particular segment that would let her identify the segment to the owner without revealing the file name to everybody in her neighbourhood. Moreover, the information possessed by Alice (the *fingerprint*) must allow her to challenge Bob before commencing the download. Bob should convince Alice that he in fact owns the requested file segment.

The underlying idea of RASH is the complete separation of fingerprint acquisition from its presentation to a prospective segment provider. If Alice wants to download a segment of a given file, she first has to obtain the requisite fingerprint. This operation is carried out in a way that preserves Alice's anonymity. In the second stage (the actual segment *content* acquisition), Alice presents the fingerprint to peers in her neighbourhood in a way that renders it useless to anyone who does not possess the file. In addition to playing the role of a secret file identifier, the fingerprint is also used as a challenge to which the prospective sender has to respond properly—to convince Alice that the offer is serious.

Fingerprint acquisition is relatively cheap (in terms of bandwidth) and safe (in terms of privacy), and can be repeated, possibly at different times and in different environments. It can also be arbitrarily separated in time and location from the segment acquisition stage. One strategy is to acquire *multiple* fingerprints of the same segment and decide on their validity. Details of such a strategy are outside the scope of this paper. Instead, we restrict our discussion to the mechanism that can be invoked as often as needed by a node to establish to its (subjective) satisfaction that it is in possession of the correct fingerprint information. Thus, before deciding to download the segment from an unknown peer, Alice can establish considerable faith in the authenticity of the fingerprint. An attack on the protocol would require a collusion of large numbers of users over a long period of time. Such collusion is difficult to carry out consistently, especially since Alice's identity is not known during the fingerprint acquisition stage.

3. Fingerprint Acquisition

Let F denote an arbitrary file and s_i its i–th segment that Alice wants to download. The first stage of RASH allows Alice to acquire one or more fingerprints of the desired segment.

First, assume that somehow Alice is able to anonymously broadcast a short query for the segment's fingerprint and receive a reply. We will explain the exact means to achieve anonymity for this transaction later in this section. Thus, Alice broadcasts the following query:

$$Q(F, i) = \{N(F), i, S_a, S_b\} ,$$

where $N(F)$ is the file name and S_a and S_b are two random bit patterns of a definite length (e.g., 64 bits) called *salts*. A node, say Carol, receiving Alice's request and, as determined by $N(F)$ and i, being in possession of the requested segment, calculates the pair of values $\{H(S_a \circ s_i), H(S_b \circ s_i)\}$, where H is a known cryptographically-secure hash function and $\circ$ denotes concatenation. In plain words, Carol computes two hash values of the segment affected by the two salts generated by Alice. Then, Carol returns to Alice her reply:

$$R(F, i) = \{S_a, H(S_a \circ s_i), H(S_b \circ s_i)\} ,$$

where the role of repeating S_a is to allow Alice to match Carol's reply to the original query. The fingerprint to be used by Alice for the actual file acquisition is:

$$P(F, i) = \{S_a, S_b, H(S_a \circ s_i)\} .$$

Before we move on to the second (segment-acquisition) stage, let us explain how the exchange between Alice and Carol can be carried out in an anonymous way. Alice has to unavoidably broadcast the request $Q(F, i)$, because it is not known who (if any) of her peers will respond. Note that obfuscating $N(F)$ in $Q(F, i)$ is not enough to prevent Mallory from discovering what file Alice seeks. For example, the hash of the file name, or even the hash of the entire file (assuming it is known via a directory lookup or pre-existing advertisement) could be obtained by Mallory just as by Alice and stored in a dictionary. A different method of preventing Mallory from associating $N(F)$ with Alice is necessary, and we present two solutions to this anonymity problem. The first is cryptographically secure but relatively complex. The second appears more practicable and is "quite secure," while it also enlarges the pool of potential fingerprint providers.

3.1. Trusted Circle of Anonymity (TCA)

Owing to the relatively low bandwidth requirements for the exchange between Alice and Carol, a cryptographically-secure solution to the anonymity problem can take advantage of the *Unconditional Sender and Recipient Untraceability Algorithm* [6]. The idea behind this algorithm is that a group of peers arrange themselves into a logical ring and, at regular intervals, adjacent pairs of peers flip coins between them, using a fair coin flip protocol that is secure from eavesdroppers [1]. After every flip, each peer announces either "same" if the coin flip with the peer on their left had the same result as the coin flip with the peer on their right; otherwise, the peer announces "different." When Alice wishes to broadcast a binary message, she inverts her announcement for each "1" she wishes to send, and does not change her announcement for each "0." Unfortunately, this protocol can be easily disrupted: Mallory need only send a meaningless stream of random announcements into the circle to permanently and untraceably disrupt all communications.

To improve this protocol, let us assume that the network in Alice's neighbourhood has two important properties: 1) an attacker cannot physically prevent communication between two chosen participants (the *inseparability property*), 2) there are at least two honest participants in the circle. The validity of these assumptions may be open for debate. However, given that all participants in a circle are within each other's transmission range and are potentially mobile, it would at least be a great challenge for Mallory to physically disrupt communication between precisely two chosen peers, and to do so while the disrupted nodes move independently. Similarly, security-conscious peers could choose to enter circles only if there is at least one member whom they trust who is also willing to enter the circle (more on this in Section 5).

Once the two postulated properties hold, we can resort to the *Fail-Stop Agreement Protocol* [9], which uses a *Fail-Stop Byzantine Agreement* and a signature scheme whereby forgery can be proved to guarantee *serviceability* and *unconditional untraceability*—that is, disrupters can be identified and removed from circles, and the untraceability of all legitimate messages sent in the circle cannot be compromised.

One issue with TCA is timing the formation of circles of peers. Circles should be formed randomly and disbanded after a certain time-out period in which no data has been transmitted on them. If instead circles were formed on–demand by nodes wishing to send fingerprint requests, then the identity of the node that is initiating the fingerprint requests is indirectly revealed. Also, a circle must have at least three members to provide any anonymity at all. Further details of the low-level protocol for forming circles are beyond the scope of this paper.

One can convincingly argue that forming TCA's is not feasible in the wireless environment. Note that a circle can only be formed by a group of peers whose connectivity graph is a clique. That is, only a subset of all nodes in the transmission range of Alice can participate in Alice's circle, which may significantly reduce the opportunities for acquiring fingerprints. Furthermore, the overhead of maintaining circles in a highly mobile environment can be prohibitive, rendering the scheme practically ineffective, especially under conditions of non-trivial mobility.

3.2. Direct Anonymous Broadcast (DAB)

The fingerprint acquisition scheme presented in this section, called DAB, is an adaptation of an idea described by Stajano [7], which states that by removing the "from" field in a broadcast packet's header, the sender can essentially remain anonymous. Even though this kind of anonymity is not mathematically provable, Stajano successfully argues that tracing a transmission to its sender by measuring the signal strength, timing, and phase of the transmitted packets is at least a difficult, inaccurate, and expensive operation. At the very least, a group of nodes would have to conspire to triangulate the sender and then make a visual identification of the person possessing the mobile device.

Technically, the sender address field cannot just be removed from the data-link frame. To play by the rules of well–formedness of frames, a source address should be present, but it can be generated at random and switched as often as needed, especially after the completion of an exchange, when the continuity of a transaction in progress need no longer be preserved. This way, the logical ID of a node can be disassociated from the data-link address, and data-link layer transmissions can be rendered practically anonymous.

In her spare time, preferably when energy is of no concern (e.g., when the node is recharging its battery), Alice generates a number of public/private key pairs to assist her in future fingerprint acquisitions. Alice's query, broadcast (anonymously) to all nodes in the neighbourhood, now looks as follows:

$$Q(F,i) = \{N(F), i, T_a, T_b, O, \mathrm{TTL}\} ,$$

where the first two items are as before, T_a and T_b are two salts of a fixed length (e.g., 64 bits), O is the public key of one of Alice's key pairs, and TTL (time to live) is set to a uniformly distributed random value in $[\mathrm{TTL}_{min}, \mathrm{TTL}_{max}]$. We then denote:

$$S_a = T_a \circ O ,$$
$$S_b = T_b \circ O .$$

When Carol receives Alice's query, she does one of two things. If she has no file matching $N(F)$, she decrements the TTL counter, and, if it is still positive, rebroadcasts the request to her neighbours. Otherwise, i.e., if TTL has reached zero, Carol does not rebroadcast Alice's request.

If Carol happens to possess a file matching Alice's description, she replies by broadcasting (anonymously):

$$R(F,i) = \{S_a, E_O(H(S_a \circ s_i) \circ H(S_b \circ s_i)), \mathrm{TTL}\} ,$$

where E_O denotes encryption with the public key O, and TTL is set to a random value in $[\mathrm{TTL}_{max}, \mathrm{TTL}_{max} + \Delta]$, for some reasonable positive Δ. The TTL setting of Carol's reply is guaranteed to be larger than Alice's initial value—to give the reply a good chance of making it all the way back to Alice.

When a node other than Alice receives Carol's message, it checks whether it has seen previously, within some time interval, a fingerprint request containing the salt S_a (or, technically, the salt T_a and public key O). If this is the case, the node will rebroadcast the reply; otherwise, it will ignore Carol's message. Alice, having recognized her own salt S_a in the reply, decrypts the hash values with her matching private key and constructs $P(F,i)$ as before.

The reason why Carol encrypts the hash values returned to Alice with the public key is to hide the fingerprint from everybody except Alice. Note that the salts applied to s_i contain O as a substring. This design (as opposed to one where $S_a = T_a$ and $S_b = T_b$) prevents playback attacks in which Mallory substitutes her own public key, O', for O and rebroadcasts Alice's query, thereby learning $H(S_a \circ s_i)$ and $H(S_b \circ s_i)$ from Carol.

In addition to this obvious precaution, Alice and Carol should play a simple trick to avoid leaking information about their special roles in this exchange. Specifically, Carol should rebroadcast the query received from Alice (if its TTL counter is still nonzero) as otherwise she may reveal to her neighbours that she is the actual fingerprint provider. Similarly, Alice should rebroadcast Carol's reply—to disguise the fact that she happens to be its recipient.

The rebroadcasting feature of this scheme, in addition to increasing the population of prospective fingerprint providers, plays the role of a poor man's mix network, further obfuscating the fingerprint acquisition. Even if a node finds out which of its neighbours sent a particular request, it does not know whether the request originated at that node, or if it was rebroadcast on its way from somewhere else.

DAB can be generalized to request fingerprints of multiple segments at once, i.e., Alice's query may look like this:

$$Q(F,I) = \{N(F), I, T_a, T_b, O, \mathrm{TTL}\} ,$$

where $I = \{i_0, \ldots, i_{k-1}\}$ is a set of k segment numbers. Carol's response to this query will include a subset of I and the corresponding sequence of hash values, i.e.,

$$R(F,J) = \{S_a, E_O(J \circ H(S_a \circ s_{i_0}) \circ H(S_b \circ s_{i_0}) \circ \ldots$$
$$\ldots \circ H(S_a \circ s_{i_{m-1}}) \circ H(S_b \circ s_{i_{m-1}})), \mathrm{TTL}\} ,$$

where $J \subseteq I$ and $m = |J|$. From this response, Alice can decode the specific fingerprint of each segment.

4. Dual Hash Exchange (DHX)

In this section we describe DHX, the Dual Hash Exchange protocol for segment transfer negotiation. Suppose that Alice needs the i–th segment of F and that she earlier acquired a fingerprint of that segment. Alice asks Bob for that segment and she wants to make sure that Bob in fact does have the segment before she decides to proceed with the download. Thus, Alice sends to Bob the fingerprint:

$$P(F, i) = \{S_a, S_b, H(S_a \circ s_i)\} \ .$$

which plays the role of the segment identifier as well as a challenge. First, Bob attempts to identify the requested segment by prepending to it S_a, evaluating the hash function H and comparing the result to $H(S_a \circ s_i)$ supplied by Alice. Having found a matching segment, Bob evaluates $H(S_b \circ s_i)$ and returns $\mu(H(S_b \circ s_i))$ to Alice as his response to the challenge, where the function μ is as described in the upcoming Section 4.5. This convinces Alice that Bob in fact does have the segment. Note that there is no trace of the file name $N(F)$ in that exchange. In fact, Bob will locate and send the segment even if his copy of the file has a different name than the one known to Alice. Also note that Mallory cannot respond to the challenge unless she has the segment s_i. As the hash value expected by Alice is affected by the random salt S_b, Mallory cannot mount dictionary attacks, which would require her to store the hash value of s_i for all possible values of the salt.

Note that, if Alice has fingerprints of multiple segments obtained for the same values of S_a and S_b, she can combine them into a single query to Bob, e.g.,

$$P(F, I) = \{S_a, S_b, H(S_a \circ s_{i_0}), \ldots, H(S_a \circ s_{i_{k-1}})\} \ ,$$

where $k = |I|$. Bob, who may only have a subset of those segments, will respond with the set of their indices, $J \subseteq \{0, \ldots, k-1\}$, and the corresponding sequence of $\mu(H(S_b \circ s_j))$. Then, Alice will download the indicated segments from Bob.

4.1. Caveats

One can see that the DHX protocol is not perfect. If Mallory is in possession of the segments sought by Alice, she will still learn what segments Alice was requesting. Furthermore, nothing prevents Mallory from sending garbage to Alice instead of the requested segments, provided that she can produce $\mu(H(S_b \circ s_i))$. However, the protocol will at least reduce the number of attackers who can send garbage to Alice, and Alice still has the two hash values to confirm that she was not sent garbage.

Faced with these issues, we may ask whether it is possible for Alice to request a file in such a way that Mallory would not know what file is being requested, *even* if she has the file. Of course, the answer to this question is no. Even were such a scheme possible, Alice would *eventually* have to reveal to her peer, be it Bob or Mallory, in which file she is interested (otherwise, her peer would be unable to transmit the file to her). Hence, any two-party P2P transfer protocol *must* suffer from this deficiency of DHX: it is eventually revealed to Mallory what file Alice wants, provided Mallory has the file, and she could send some small amount of garbage to Alice before being detected.

4.2. Efficiency versus Privacy

At first sight, it may seem that the fingerprint acquisition scheme, be it TCA or DAB, could also be used for downloading segments, thus preserving the anonymity of Alice and Bob, unlike the DHX protocol. Unfortunately, neither of the two fingerprint acquisition schemes presented in Section 3 is equipped to cater to high-volume data transfers. This is certainly true in the case of TCA and it is also the case in DAB. Indeed, DAB relies on “blind” rebroadcasting, which engages a significant number of bystanders into transactions that, from their point of view, are useless. On the other hand, with DHX, it is possible for Bob to transmit a segment to Alice at the same time that Dave is transmitting a segment to Carol (provided that their wireless transmissions do not interfere).

The maintenance of TCA’s as well as negotiations via DAB result in relatively low effective throughput and large delays. Consequently, while useful for the anonymous acquisition of short fingerprints, they are grossly inadequate for transmitting potentially large file segments. Therefore, the two types of transactions, i.e., fingerprint acquisition and segment transfers, should be viewed as two conceptually different modes of communication with different goals.

4.3. Segment Search Cost

DHX, as presented in Section 4, involves a rather high cost of searching through segments on Bob’s end. Bob has to scan through his entire collection, hashing all his segments with the salt S_a. This means that Bob will end up hashing all the bytes of all his files to arrive at a negative answer, and about 1/2 of them on the average to locate a segment that he actually owns. Fortunately, it is possible to drastically reduce the cost of the segment search operation. Suppose that, as stored in Bob’s repository, the segments (in addition to being naturally searchable via file names) are internally hashed by short (e.g., 32-bit) keys $\xi(s)$ which give a statistically good (albeit not necessarily perfect) conflict avoidance. One possible candidate for ξ is simply H, although ξ can be shorter: its role is not to identify segments but to act as a traditional hash function facilitating searching among objects that are uniquely identified only by themselves.

The modification involves a small addition to the fingerprint information. The response $R(F,i)$ sent anonymously by Carol to Alice in the fingerprint acquisition stage includes one extra item, namely $H(S_a \circ \xi(s_i))$, i.e., the salted hash of the search key of the requested segment. For a multiple segment response, $R(F,J)$, the extra item occurs for every segment in J. Similarly, the fingerprint sent by Alice to Bob in the DHX stage is augmented by $H(S_a \circ \xi(s_i))$, for each segment s_i that Alice wants to acquire from Bob. Now, instead of painstakingly hashing entire segments, all Bob has to do to find the segment sought by Alice is to apply H only to the search keys ξ of his segments, prepending the salt S_a to every key. Note that the security of this scheme, from the viewpoint of thwarting dictionary attacks, depends on the size of the salt S_a, not on the size of the hash key used for locating segments. Bob resolves possible conflicts by evaluating $H(S_a \circ s_i)$ on the segments pointed to by the key.

4.4. Public/Private Keys in DHX

Significant enhancements to the privacy and resilience of DHX are possible, if we assume that the nodes are equipped with (preferably certified) public/private key pairs. Those keys could be set up, hardwired, and certified by the device manufacturer, or created by the user and certified by temporarily plugging the device to a solid infrastructure network, i.e., the Internet.

First of all, with the keys in place, Alice could carry out the segment download negotiation, as well as the actual download, over a channel encrypted with a symmetric key established via the Station-To-Station protocol [6]. Second, the protocol may force Alice to sign her requests with her certified private key—to enforce fairness and eliminate sleep deprivation attacks [7] against Bob. Bob could recognize multiple frequent requests arriving from the same source and impose a minimum time separation between their service. Specifically, along with her request, $P(F,I)$, Alice would be obliged to send to Bob $\{X, D_{Alice}(H(P(F,I)) \oplus X)\}$, where X is a random string of bits, $\oplus$ is exclusive or, and D_{Alice} is encryption with Alice's private key. What is important about Alice's certificate in this context is that she cannot easily generate a new public/private key pair (effectively changing her identity), and issue another request to Bob before her time-out period is over.

Alice combines her signature with the application of X to the hash of her request because she wants to prevent Bob from being able to demonstrate to other parties that Alice has issued that particular request. If Alice simply signed the hash of her request, Bob could hold on to it and then prove to everybody that Alice once queried him for segment s_i (though he would need to have s_i to prove his claim). With X in place, Bob is still able to verify that the request is coming from Alice, but Alice can easily repudiate any claims that she ever issued that request. This is because she can always produce an innocuous request P_{Fake} and claim that the random bit stream she used with Bob was not X, but rather $Y = X \oplus H(P(F,I)) \oplus H(P_{Fake})$.

However, one remaining flaw with this scheme is that given $P(F,I)$, $D_{Alice}(H(P(F,I)) \oplus X)$, and X, Bob can impersonate Alice. For any request, P_{Bob}, Bob need only compute $Z = X \oplus H(P(F,I)) \oplus H(P_{Bob})$. Then, taking advantage of the fact that:

$$H(P(F,I)) \oplus X = H(P_{Bob}) \oplus Z$$
$$\Rightarrow D_{Alice}(H(P(F,I)) \oplus X) = D_{Alice}(H(P_{Bob}) \oplus Z)$$

Bob can make it appear that his request is actually coming from Alice by attaching $D_{Alice}(H(P(F,I)) \oplus X)$ and Z to his request.

The solution to this vulnerability requires both parties involved in the request protocol to contribute to the creation of the random bit stream. Specifically, Alice generates a random bit stream, X_{Alice}, and Bob generates another bit stream, X_{Bob}. After the two parties exchange their random bit streams over the encrypted channel, the value of X used for the session is:

$$X = X_{Alice} \oplus X_{Bob}$$

Provided that both parties commit to their random bit streams before exchanging them (several commitment schemes are discussed by Schneier [6]), Bob will be unable to select X_{Bob} in a way that would allow him to impersonate a peer who previously sent him a request.

4.5. The Function μ

Recall from Section 4 that when Alice challenges Bob by presenting $H(S_a \circ s_i)$, Bob must correctly respond with $\mu(H(S_b \circ s_i))$, for some as-yet undefined function μ.

First, note why it is important that a function μ be applied to Bob's reply. Were Bob to simply reply with $H(S_b \circ s_i)$, there would be an insecurity in the system. Imagine that Bob does not possess s_i. In theory, he should be unable to reply to Alice's challenge. However, were Bob to replay Alice's challenge to Carol, who has s_i, Bob would learn the proper response to the challenge from Carol. He could then replay the response to Alice, convincing her that he possesses a segment that he actually does not.

A simple solution to this vulnerability is to define, for any string of bytes z:

$$\mu(z) = M_X(z) \ ,$$

where X is the random bit stream established by Alice and Bob at the beginning of their DHX session (as described in Section 4.4), and M is a MAC function (a keyed hash function). This definition of μ ensures that the proper reply to a given DHX challenge is unique each time the challenge is made, thereby thwarting the aforementioned class of replay attacks.

4.6. Reciprocity Enforcement

One can think of implementing a reciprocal file exchange policy on top of DHX. First, Alice sends her request P_{Alice} to Bob, and, if there is an item in P_{Alice} that Bob can provide, he replies by sending his request, P_{Bob}, to Alice. If Alice finds a segment in P_{Bob} that she can provide, the transfer begins.

However, note that there is no way to ensure that Bob transmits his file to Alice after she transmits hers to him (or vice versa, if they transmit in the opposite order). The simplest step towards avoiding this kind of abuse is to use a cryptographically-secure coin flip [6] to determine which peer should go first. This reduces the possibility that one peer will be able to receive a file without transmitting to 50%, and ensures that the exchange is fair. For instance, if the communication link fails for non-malicious reasons half-way through the exchange (e.g., due to congestion or interference), there is an equal probability that either peer will have the file he or she desired.

5. Trust Models

The fingerprint acquisition schemes presented in Section 3 introduce overheads which we would rather avoid if in contact with trusted parties/peers. Globally unique, certifiable identifiers, e.g., public keys, can be used for this purpose. Alice may, when in Bob's communication range, add his ID to her list of trusted peers. Bob will be informed, and, if he chooses to trust Alice, these two peers will have established mutual trust. If Bob chooses not to trust Alice, his ID will be removed from her list of trusted peers. When Alice then encounters Bob in the future, she can ask him to prove his identity. Then, for the remainder of the time they spend in each other's transmission range, Alice will treat Bob as a trusted peer. This means that Alice:

- will communicate directly with Bob over an encrypted channel (using a shared key generated by the Station-To-Station protocol) to request file fingerprints,
- will eagerly enter any circle together with Bob (assuming TCA fingerprint acquisition),
- will be able to start requesting file segments from Bob immediately after receiving a fingerprint.

Further shortcuts are also possible, e.g., if Alice receives a fingerprint from a source other than Bob, she need only confirm with Bob that the hash value of his segment matching $\{N(F), i\}$ is equal to the fingerprint value she received in order to download the segment from Bob. Alice can also expand her list of trusted peers by asking Bob to provide her a (possibly partial) list of peers whom he trusts, as well as a digitally-signed document stating that Bob trusts Alice. If Bob chooses to provide this information to Alice, Alice may then repeat the same steps for establishing mutual trust with the peers whom Bob trusts.

This trust model, though useful in some contexts and minimal in overhead, is not scalable to large networks where peers almost certainly download files not owned by their trusted friends. Hence, we will next consider two scalable trust strategies that can be applied in the absence of pre–existing trust relationships or a central trust authority.

5.1. Swarm Intelligence

A trust model based on *Swarm Intelligence* [2] could be adapted to work with RASH. In this model, peers decide whether to trust each other based on evidence about their past behaviour available in the network. To implement Swarm Intelligence in RASH, after each reciprocal file exchange, the peer who transmitted first (Alice) could sign a *testimonial* regarding Bob's eagerness to hold up his end of the bargain. To deal with the possibility that Bob was unable to transmit because of congestion, Alice should sense the medium before generating her testimonial. If the medium is congested, she should generate no testimonial about Bob, so that he will be neither penalized nor rewarded for factors beyond his control. Of course, Bob could transmit junk bits to make the medium appear congested, thereby freeing himself of any obligation to transmit a file segment to Alice; however, Alice could sense the medium for long enough to make Bob's deception at least as expensive (in terms of used energy) as actually sending the segment to her.

Testimonials would then circulate in the network. Later, when Bob attempts to initiate an exchange with Carol (or, when Carol is contemplating an exchange with Bob), she could request a review of Bob from her peers. Using the received information, and taking into account the trustworthiness of whoever signed the testimonials, Carol could decide whether initiating a transfer with Bob is worth the risk. Unfortunately, it could be a very expensive and slow process for Carol to request all available documents on each person with whom she wishes to initiate a transfer. A more efficient approach may be needed.

5.2. An Adaptation of URSA

URSA [5] is intended as a protocol for access control in mobile ad-hoc networks. In URSA, a node must present a ticket that has been signed by its neighbours in order to have its packets forwarded on the network. Only nodes that behave properly will have their tickets renewed by their neighbours.

Under this scheme, Carol will assume that Bob is a trustworthy peer if he is in possession of a valid ticket. As before, after the transfer is complete, the node that transmitted first signs and distributes a testimonial about its peer. Such a testimonial has an expiration time E. If the validity period of a ticket is T_{cert} and the time at which the testimonial is signed is D, then:

$$E = D + T_{cert} + T_{const}$$

for some global constant $T_{const} \geq 0$. This formula guarantees that the documents will persist in the network for at least one of the subject's ticket renewals. It should be noted that no synchronized clocks are needed for this scheme to work. The clocks at different nodes merely need to be close enough for Bob's neighbours to detect any blatant cheating, i.e., distributing a document about Carol with $E = D + T_{cert} + T'_{const}$ for $T'_{const} >> T_{const}$.

When Bob's ticket nears expiration, he requests that his neighbours renew it for him. At this point, his neighbours collect all the testimonials on Bob available to them from the network. Using the information contained in these testimonials, they determine if Bob is trustworthy enough to remain in the game. If he is, his ticket is renewed. Otherwise, Bob will be unable to initiate any transfers without a valid ticket—at least until all the negative testimonials about him expire, and he is given a fresh chance to join the network. With this approach, the expensive task of collecting testimonials on Bob need only be performed when his ticket comes up for renewal.

6. Conclusions

We have introduced a protocol for anonymous file exchange in a mobile ad-hoc wireless environment. The protocol strives to preserve the anonymity of peers while trying to avoid high complexity, especially in its bandwidth-intensive parts, i.e., actual segment transfers. There are several open issues with our design. First, even though Mallory cannot determine the correct response to Alice's challenge by listening to her fingerprint request and Bob's reply, she can still modify the communication in transit without any consequences. Similarly, Mallory could transmit a meaningless reply to Alice's request. Alice's only defence is to take any information she receives over the anonymous channel with a grain of salt (note the pun), and to re-request file information if she believes the information she received was incorrect (for example, if she cannot find any peers with a file matching the received fingerprint). One strategy is to re-acquire the file fingerprint information at different points in time and space before forming segment acquisition requests.

Another issue with RASH is the linear complexity of segment lookups performed by Bob in response to Alice's download queries. In Section 4.3, we showed how to carry out those lookups by searching among short keys representing the segments, but the best we can do at the moment is to scan those keys linearly (the reason being that a hash function must be applied to each of them before a comparison). We are currently studying this problem and plan to address it in the next version of RASH.

References

[1] M. Blum. Coin flipping by telephone: A protocol for solving impossible problems. In *Proceedings of the 24th IEEE Computer Conference (CompCon)*, pages 133–137, 1982.

[2] L. Eschenauer, V. D. Gligor, and J. Baras. On trust establishment in mobile ad-hoc networks. Technical report, University of Maryland, 2002.

[3] P. Gole and D. Boneh. Almost entirely correct mixing with applications to voting. In *Proceedings of 9th ACM conference on Computer and Communication Security*, pages 59–68. ACM, 2002.

[4] M. Jakobsson. A practical mix. In *Proceedings of EUROCRYPT 1998*, pages 448–461. Springer-Verlag, LNCS 1403, 1998.

[5] H. Luo, J. Kong, P. Zerfos, S. Lu, and L. Zhang. URSA: Ubiquitous and robust access control for mobile ad-hoc networks. *IEEE/ACM Transactions on Networking*, October 2004.

[6] B. Schneier. *Applied Cryptography, Second Edition: Protocols, Algorithms, and Source Code in C*. John Wiley & Sons, Inc., 1996.

[7] F. Stajano. *Security for Ubiquitous Computing*. John Wiley & Sons, Ltd., February 2002.

[8] P. Syverson, D. Goldschlag, and M. Reed. Anonymous connections and onion routing. In *Proceedings of the 18th IEEE Symposium on Security and Privacy*, pages 44–54. IEEE, 1997.

[9] M. Waidner and B. Pfitzmann. The dining cryptographers in the disco: Unconditional sender and recipient untraceability with computationally secure serviceability. In *Proceedings of EUROCRYPT 1989*, page 690. Springer-Verlag, LNCS 434, 1990.