Studienarbeit

Concepts of Anonymous Reputation Management

Henryk Plötz
ploetz@informatik.hu-berlin.de

WS 2006/2007
February 29, 2008 (Revision: 1.14)
Humboldt-Universität zu Berlin
Institut für Informatik

Contents

1 Introduction

1.1 Definitions

1.1.1 Anonymity

Anonymity (of an entity) refers to the property of that entity of not being identifiable within a set –the so-called ‘anonymity set’– of similar entities.

Typically anonymity is an all-or-nothing property, e.g. one can’t be just a little bit anonymous. In order to be able to differentiate between different anonymization systems and the strength of their anonymity guarantees several measures can be applied. The most common is the *anonymity set size* (first used in [Chaum, 1988]): If there are two system with four users each, and the first system has an anonymity set of size four and the second system has two anonymity sets of size two each, then obviously the first system is to be preferred.

On a side note: For law enforcement purposes in constitutional states it’s sufficient for the anonymity set size to be greater than or equal to 2 (of course including possible circumstantial reductions of the set size, e.g. the remaining members must not have alibis). For different purposes the anonymity set size might not be the best metric.

Consider again a system with four users and an anonymity set of size four, with respect to committing a particular action. If one of the members committed the action with probability 0.91 and the other three members with probability 0.03 each, then it’s clearly not a particular good system. Under certain circumstances and regimes this probability distribution might be enough to lead to serious consequences for that particular member with the high probability.

This leads to a different –information theoretic– anonymity metric as given in [Serjantov and Danezis, 2002], the *effective size of the anonymity probability distribution* $\mathcal{S}$ is defined to be the entropy of that distribution. From its definition it’s clear to see that $0 \leq \mathcal{S} \leq \log_2 |\Psi|$ (where Ψ is the anonymity set). A value of $\mathcal{S} = 0$ means that the system offers no anonymity and a particular user is clearly identified. A value of $\mathcal{S} = \log_2 |\Psi|$ is the maximum value and means perfect, equally distributed anonymity for all users.

2 Reputation Systems

Reputation systems are usually collaborative systems where users rate each other, for example based on their contributions. These ratings are then somehow integrated to assign each user one or more ‘reputation’ values, based on that user’s conduct in the past. The reputation value can be useful as a predictor either for that user’s likely future conduct, or for contributions by that user which have not yet been rated.

The addition of reputation systems can introduce accountability into peer-to-peer systems and thus help to discourage misbehaviour and/or encourage good behaviour.

2.1 Eigentrust

The Eigentrust reputation system has been described by Sepandar D. Kamvar, Mario T. Schlosser and Hector Garcia-Molina in their 2003 paper “The EigenTrust algorithm for Reputation Management in P2P Networks” ([Kamvar et al., 2003]).

The main idea behind Eigentrust is the notion of transitive trust: If peer X trusts peer Y and peer Y trusts peer Z then peer X most likely also trusts peer Z at least to some extent. This is modeled on the observation that real-life trust relationships are usually transitive, though with reduced strength as the number of intermediaries grows. It is worth noting that relationships other than trust relationships are not generally transitive, so the Eigentrust model will not be perfectly suited for some reputation systems (depending on which types of relationships these systems assign ratings to).

The Eigentrust algorithm starts with each peer i computing local trust values s_{ij} for all peers j that it has ratings for. The original formula for the calculation of these trust values is

$$s_{ij} = \mathrm{sat}(i,j) - \mathrm{unsat}(i,j)$$

where $\mathrm{sat}(i,j)$ and $\mathrm{unsat}(i,j)$ give the number of satisfying and unsatisfying transactions that peer i had with peer j, respectively. These values are then normalized to a maximum of 1 and cropped to be greater than or equal 0:

$$c_{ij} = \frac{\max(s_{ij}, 0)}{\sum_j \max(s_{ij}, 0)}$$

Define matrix $C =: [c_{ij}]$ and the vectors $\overrightarrow{t}_i$ as the values t_{ik} and $\overrightarrow{c}_i$ as the values c_{ij} then $\overrightarrow{t}_i$ can be calculated as $\overrightarrow{t}_i = C^T \overrightarrow{c}_i$ and is a trust vector for all peers based on peer i's trust and the weighed transitive trust of all of peer i's direct acquaintances.

When this is iterated even further as

$$\overrightarrow{t}_i^n = (C^T)^n \overrightarrow{c}_i$$

for a large n it will eventually converge to some value $\overrightarrow{t} = (C^T)^n \overrightarrow{c}_i$ which is the global trust value for the trust network. For specific reasons this will even converge to the same trust vector $\overrightarrow{t}$ independent of which trust vector $\overrightarrow{c}_i$ was originally started with, namely the left principal Eigenvector of the matrix C. (Hence the name Eigentrust.)

3 ARM4FS

3.1 System Components

The ARM4FS system consists of several system components: the Storage Provider and the Anonymous Reputation Provider which in turn consists of the Anonymization Layer and the Reputation System.

3.1.1 Storage Provider

The Storage Provider is responsible for the actual storage of the files in the system. Several standard mechanisms exist for this purpose and the actual storage method is not really important. In the following an HTTP based implementation will be used, where a standard HTTP file server provides storage for the files and associated meta-information.

3.1.2 Anonymous Reputation Provider

The Anonymous Reputation Provider is the composite system that keeps track of – and performs calculations on– the user's reputation information. It is comprised of two subsystems, in order to facilitate easy replacement of the reputation function being used.

Anonymization Layer

The Anonymization Layer is the part of the Anonymous Reputation Provider that is communicating with the outside world. Its purpose is to safeguard the anonymity of its users by restricting its output information. The Anonymization Layer also handles all user management tasks.

Reputation System

The Reputation System is the sub-component that is actually responsible for the calculation of the reputation values. It pseudonymized feedback values from the Anonymization Layer sub-component and then answers queries for reputation values that are mediated by the Anonymization Layer.

Client

The Client is what the users actually use. With it the users can upload/download/list files to/from/on the Storage Provider, query reputation values and send feedback.

3.2 Data Structures and Procedures

3.2.1 Data Structures

In order to keep track of participant and resource information the system needs to manage a few data structures which will be described in the following subsection.

ObjectKey An *ObjectKey* is a public identifier that uniquely identifies a resource through an embedded hash representation of that resource (or a description thereof). For obvious reasons the hash must have been generated with a cryptographically strong hashing algorithm, such as SHA-1.

The ARM4FS system uses the general form of an URI for its ObjectKeys and for the example of ARM4FS a SHA-1 hash will be appended as a query string like so: `http://`*`sp`*`/get?h=`*`hash`*.

UserId A *UserId* identifies a user of the system through his or her asymmetric public/private key pair and is generally not public. The private key will always be kept private at the user's client while the 'public' key will only be used in a secure session to the Anonymization Layer and not be available to the general public.

The term 'UserId' generally refers to the asymmetric key pair, even though the server will of course only possess the public key part.

UserPseudonym A *UserPseudonym* is a randomly generated identifier that serves as a pseudonym for the user's identity when it is bound to a resource through an AuthorTag. A new UserPseudonym is generated for each user/resource combination (aka AuthorTag), and it is important that different UserPseudonyms of the same user can not be correlated to each other. Therefore they must actually be generated through a cryptographically strong random number generator–and not for example by salting and encrypting the UserId.

Basically the UserPseudonym acts as a key for easier database access. It also means that the Anonymous Reputation Provider does not have to store all the ObjectKeys for each user.

AuthorTag An *AuthorTag* establishes authorship for a specific user on a resource and is a (public) signed (by the Anonymous Reputation Provider) tuple consisting of a UserPseudonym and an ObjectKey. A new UserPseudonym is generated each time a new AuthorTag is created, and then internally (to the Anonymization Layer) associated with the UserId.

AuthorTags are represented as empty XML elements of name `AuthorTag` with three attributes: `UserPseudonym`, `ObjectKey` and `signature`. The signature is a standard PKCS#1 padded RSA SHA-1 signature over the string *`ObjectKey?UserPseudonym`*.

3.3 Protocols

3.3.1 Client to Storage Provider

The client and the Storage Provider converse through a standard, REST-like, HTTP-based protocol.

Uploading Files

In order to upload a file the client issues an HTTP PUT request to `http://`*`sp`*`/put` and transmits the file that is to be uploaded. The Storage Provider then answers with an HTTP `201 Created` status code and a `Location` header. This `Location` header indicated the new URI under which the file later will be accessible. This URI also is the ObjectKey of that file.

It is expected that the file can not be retrieved until a user has established authorship on it, and that unowned files will be deleted after some time.

Establishing Authorship

After retrieving an AuthorTag for the new file from the Anonymous Reputation Provider the client will issue an HTTP POST request to `http://`*`sp`*`/storePermanent` and submit the ObjectKey and AuthorTag to the Storage Provider (in the body of the POST request, with a Content-Type of `application/x-www-form-urlencoded`).

The Storage Provider then takes the necessary steps to permanently store the file and notes the AuthorTag in its meta-data for that file, to be delivered in all file listings containing that file.

Listing Available Files

In order to retrieve a list of available files the client issues an HTTP GET request to `http://`*`sp`*`/xml_index` which will yield an XML formatted listing of the files on that Storage Provider.

The document element is of name `files` with no attributes. This contains several `file` elements (no attributes) which each in turn have several child elements whose contents describe properties of the file they pertain to:

time A float containing the upload time as seconds (with fraction) since epoch (January 1st, 1970).

name A server generated name to describe the file. For example this might be generated from MP3 ID3 tags.

hash A SHA-1 hash of the contents of that file.

type The mime-type of that file, guessed by the server through the standard file(1) command.

size The size of that file.

ObjectKey The ObjectKey of that file, also serves as the URI to retrieve the file.

AuthorTag The AuthorTag of that file, as generated by the Anonymous Reputation Provider. Differing from the other child elements in that it is an empty element where all the data is in the attributes (see above).

3.3.2 Client to Anonymous Reputation Provider

All communications between the client and the Anonymous Reputation Provider takes place within the context of a ‘session’. Each session is an authenticated and encrypted channel, using SSL/TLS. This provides for end-to-end encryption between both parties and also authenticates the Anonymization Layer component through its server certificate.

For the authentication of the client there were two choices: the use of SSL client certificates, or some custom mechanism. The SSL client certificate mechanism would have had the advantages of being relatively well tested and having multiple interoperable implementations. However, there would have been the disadvantages of added complexity, and the impossibility of implementation in J2ME. Since a goal was implementing the protocol on a mobile phone in Java, and therefore with the heavily restricted J2ME subset of the Java features, SSL client certificates were out of the question because a J2ME midlet has no way of accessing the phone’s client certificate store which the phone uses to establish SSL connections.

Instead a simple, home-brewn challenge-response mechanism was chosen. After establishing the encrypted channel and authenticating the Anonymous Reputation

Provider, the client can request a challenge from the ARP and then send back a signature over the challenge to authenticate itself, thereby logging in.

A session thus is always in one of three states: *unauthenticated*, *authentication-in-progress* and *authenticated*. In the unauthenticated state only a few commands are allowed (e.g. those commands to start the authentication process). The authentication-in-progress state is entered by requesting authentication and exited through successful authentication (or abortion of the session in the case of unsuccessful authentication). In the authenticated state, finally, all regular commands are available.

The protocol inside the encrypted session is a simple line-based request-response protocol. All commands and all answers are sent on one line each and end with a new-line character. Each command is answered with exactly one response (except if the channel is closed, abnormally). Only the client can send commands and therefore only the Anonymous Reputation Provider can send responses.

Each command is a line with at least one word (without any whitespace) specifying the command to be executed. The command may optionally be followed by a list of arguments, separated by exactly one whitespace each. Neither command nor response may contain any line-break characters except for those that signal the end of the command or response.

Each response starts with a smiley face that indicates success or failure of the command. A 'successful' response starts with a happy smiley ("`:-)`", that is: the three characters colon, hypen-minus and right parenthesis) and an 'unsuccessful' response starts with an unhappy smiley ("`:-(`", the three characters colon, hypen-minus and left parenthesis). The unhappy smiley on an unsuccessful response is then followed by a space, a three-digit error code and an optional human readable error message.

The arguments of some of the commands and some of the responses inherently are binary in nature (and thus not safe for a protocol relying on whitespace and line breaks). These are defined to be transmitted using BASE64 encoding. The affected arguments or responses are: The challenge, the response and the public key of the client. Inside of the BASE64 encoding the public key is represented in ASN.1 DER encoding.

The remaining arguments or responses need no special attention: reputation and feedback information are string representation of numbers, the ObjectKey is an URL and may thus not contain most special characters (including whitespace) and the XML representation of the AuthorTag can be created without newlines.

Authentication The Anonymous Reputation Provider is authenticated through the use of an SSL server certificate, as mentioned above. In order to use this type of authentication the client must know the server certificate in advance, or at least the server's public key, or the server certificate must be signed by a trusted CA. This is a common requirement for services using SSL and of no further significance.

The client is authenticated through its UserId –which is a public/private key pair, as detailed above– by virtue of a simple challenge-response protocol. The server generates a random challenge that is sent to the client and the client responds with a signature for that challenge. The signature scheme is standard PKCS#1 SHA-1 RSA.

In order to guarantee correct authentication the commands in the authentication process must be executed exactly as specified and no deviation is allowed. No other command may be sent after sending the `login` or `create` commands and before sending the `authenticate` command. Any violation of the protocol results in immediate abnormal termination of the session. (The session also terminates when a certain time has passed after the `login` or `create` commands with no `authenticate` command received.)

User Management As has already been said each user is solely identified by a public/private key pair called the UserId. This key pair is generated by the client and the private part is never disclosed to an external entity. In order to register to an Anonymous Reputation Provider the client sends a `create` command with the public key as argument and is then given a challenge in the response. The client then must prove its knowledge of the private key by computing the right response and sending it in an `authenticate` command. When this protocol is followed successfully, the Anonymous Reputation Provider takes all necessary steps to create the new user in its internal database(s) and the session is in the authenticated state.

Logging in to the Anonymous Reputation Provider works in a similar way, but with the `login` command instead of `authenticate` (and no new user entry needs to be created at the server).

Commands and Responses

This section details the commands and responses that are exchanged in the encrypted session between the client and the anonymous reputation provider which is acting as the server. For each command a short synopsis of the command and response format is given, followed by an explanation of the arguments, the state(s) that this command is allowed in, a description of the command and its semantics and a list of the possible positive or negative responses.

`authenticate`

Synopsis `authenticate` *`response_value`*
→ *`welcome_message`*

Arguments *`response_value`* The response to the challenge, e.g. the challenge signed with the client's private key.

State authentication-in-progress only

New State authenticated

Description This command completes the authentication procedure. It must be sent immediately after receiving the response to the `login` or `create` command. The ***`response_value`*** parameter must be a BASE64 encoded representation of a PKCS#1 padded SHA-1 RSA signature over the challenge that has been received as the response of the preceding `login` or `create` command in a BASE64 encoded form. (Note that the challenge is to be signed, and *not* the BASE64 encoded representation of the challenge.)

Response

pos. *`welcome_message`* An optional server welcome message. This may be empty.

neg. None. In case of any error or protocol violation the session will be terminated.

`create`

Synopsis `create` *`pubkey`*
$\rightarrow$ *`challenge`*

Arguments *`pubkey`* The client's public key, e.g. the public key part of the UserId. The key must be given as a BASE64 encoded DER encoded RSA key.

State unauthenticated only

New State authentication-in-progress

Description This command initiates the authentication procedure for a new user. The server will respond with a challenge that the client must sign and then respond with an `authenticate` command.

Response

pos. *`challenge`* The challenge, which is a BASE64 encoded, opaque, random string.

neg. None. In case of any error or protocol violation the session will be terminated.

`login`

Synopsis `login` *`pubkey`*
$\rightarrow$ *`challenge`*

Arguments *`pubkey`* The client's public key, e.g. the public key part of the UserId. The key must be given as a BASE64 encoded DER encoded RSA key.

State unauthenticated only

New State authentication-in-progress

Description This command initiates the authentication procedure for an already existing user.. The server will respond with a challenge that the client must sign and then respond with an `authenticate` command. If the `authenticate` command is performed successfully then the server will have created the new user account and it can be used with the `login` command in the future.

Response

- **pos.** *`challenge`* The challenge, which is a BASE64 encoded, opaque, random string.
- **neg.** None. In case of any error or protocol violation the session will be terminated.

`createAuthorTag`

Synopsis `createAuthorTag` *`ObjectKey`*
→ *`AuthorTag`*

Arguments *`ObjectKey`* The ObjectKey that the server should create an an AuthorTag for.

State authenticated only

Description This command asks the server to create an AuthorTag for a given ObjectKey. The returned AuthorTag is represented as an XML snippet in the form that was defined in 3.2.1.

Response

- **pos.** *`AuthorTag`* The AuthorTag that has been created.
- **neg. 409** There already is an AuthorTag associated with this ObjectKey but for another user.

`getReputation`

Synopsis `getReputation` *`AuthorTagOrTags`*
→ *`reputations`*

Arguments *`AuthorTagOrTags`* The AuthorTag that the reputation should be retrieved for. May also be a concatenation of multiple AuthorTags in order to reduce the number of round-trips necessary for the retrieval of a large number of reputation values.

State authenticated only

Description This command lets the server retrieve the reputation values that are associated with a list of AuthorTags.

Response

- **pos.** *`reputations`* The reputation values as a list of string representations of floating point numbers, separated by spaces.
- **neg.** **403** The AuthorTag was not created by this anonymous reputation provider.

  404 The AuthorTag is not assigned to a reputation value on this anonymous reputation provider.

Important note: If multiple reputation values were requested and any of the queries fails then a negative response will be returned and it is not possible to tell which of the queries failed.

`submitFeedback`

Synopsis `submitFeedback` *`AuthorTag feedback`*
$\rightarrow \emptyset$

Arguments *`AuthorTag`* The AuthorTag of the resource that feedback is given on.

`feedback` The feedback that is being given, as a string representation of an integer of either `+1` or `-1`.

State authenticated only

Description This command submits user feedback on a particular resource to the server.

Response

- **pos.** None.
- **neg.** **403** The AuthorTag was not created by this anonymous reputation provider.

  404 The AuthorTag is not assigned to a reputation value on this anonymous reputation provider.

3.3.3 Anonymization Layer to Reputation System

As the Anonymous Reputation Provider consists of two parts –the Anonymization Layer and the Reputation System– there is a need for these to parts to communicate. Once again this is done through a simple line-based text protocol over TCP. This time encryption is not necessary, because this is internal communication on the local host.

This approach offers great flexibility and makes it possible to code the Anonymization Layer and the Reputation System in two different programming languages. It also decouples the Reputation System so that it is possible to replace the Reputation System with a different one.

For this protocol the Reputation System acts as a TCP server on port 4444 on localhost. All commands are sent by the Anonymization Layer, and all responses are sent by the Reputation System. Commands and responses are one line each, ending with a single newline character.

`newuser` *`n`* Indicates that a new user with index *`n`* has been added. There is no response from the Reputation System.

`feedback` *`n m x`* Indicates that user *`n`* sent a feedback regarding user *`m`* with value *`x`*. There is no response.

`kcabdeef` *`n m x`* Indicates the revocation of a previous feedback command. No response.

`query` *`n m`* Queries the Reputation System for the reputation of user *`m`* as seen from user *`n`*'s perspective. Response:

> *`y`* The reputation value of *`m`* from the perspective of *`n`*.

`age` *`a`* This command is used to create ageing in the reputation data by multiplying all previous feedback values with *`a`*, thereby creating a moving average. No response

`n` This command is used by the Anonymization Layer to query the current N, e.g. the *`n`* from the last `newuser` command that was seen by the Reputation System. This is needed by the Anonymization Layer after startup to determine the value that should be sent in the next `newuser` command. Response:

> *`n`* The N.

In this protocol *`n`* and *`m`* are string representations of integers from the set $\{1, \ldots, N\}$ (where N is the last *`n`* sent in a `newuser` command), *`x`* is from the set $\{-1, +1\}$ and *`a`* and *`y`* are string representations of real numbers (using the C language format string `g` or `G`).

4 Challenges

4.1 Sybil Attack

A Sybil Attack, first described by John Douceur in [Douceur, 2002], is a common attack pattern on peer to peer systems, and especially against their reputation systems. The attacker creates multiple accounts or identities that are all under his control. He can then gain advantages over other users of the system by using this network of entities, e.g. to give particularly high ratings to the attacker's contributions. This can generally not be prevented without a central authority that guarantees that distinct accounts are being held by distinct entities.

In the context of ARM4FS this need can be catered to with the inclusion of an Identity Provider into the process. The Anonymous Reputation Provider would then be modified to require either an authenticated guaranteed unique identification –which might be problematic because this would likely involve personal details of the user– or at least some external proof that the user in question does not already have an account with the ARP.

4.2 Side Channels

Side channel attacks are an attack class that can compromise the anonymity offered by the system, through the analysis of meta data and other obvious and non-obvious. This includes for example the author information from office documents, encoder comments from music files or user information from tar archives. As a counter-measure these can be filtered out automatically at least some of the time. The user must of course take care not to release documents that lead directly to their identity, e.g. printed on letterhead.

A twist on this attack type are linking attacks where different documents from the same contributor –where at least one was released with identifying information, willful or not– are being linked together to the same author to reveal the source of one of them to be the source of all of them. These have been take care of in the implementation by not giving out the author's ID explicitly and obscuring implicit hints at which documents might have been submitted by the same user. The latter is achieved by quantizing the returned reputation values to e.g. one of six possible values, thereby creating six anonymity sets.

5 Bibliography

[Chaum, 1988] Chaum, D. (1988). The dining cryptographers problem: unconditional sender and recipient untraceability. *J. Cryptol.*, 1(1):65–75.

[Douceur, 2002] Douceur, J. R. (2002). The sybil attack. In *IPTPS '01: Revised Papers from the First International Workshop on Peer-to-Peer Systems*, pages 251–260, London, UK. Springer-Verlag.

[Kamvar et al., 2003] Kamvar, S. D., Schlosser, M. T., and Garcia-Molina, H. (2003). The eigentrust algorithm for reputation management in p2p networks. In *WWW '03: Proceedings of the 12th international conference on World Wide Web*, pages 640–651, New York, NY, USA. ACM.

[Serjantov and Danezis, 2002] Serjantov, A. and Danezis, G. (2002). Towards an information theoretic metric for anonymity.

Humboldt University Berlin
Computer Science Department
Systems Architecture Group

Rudower Chaussee 25
D-12489 Berlin-Adlershof
Germany

Phone: +49 30 2093-3400
Fax: +40 30 2093-3112
http://sar.informatik.hu-berlin.de

Concepts of Anonymous Reputation Management

HU Berlin Public Report
SAR-PR-2008-08

Februar 2008

Author(s):
Henryk Plötz

Reports published by Humboldt University Berlin, Computer Science Department, Systems Architecture Group.

1. SAR-PR-2005-01: Linux-Hardwaretreiber für die HHI CineCard-Familie. Robert Sperling. 37 Seiten.
2. SAR-PR-2005-02, NLE-PR-2005-59: State-of-the-Art in Self-Organizing Platforms and Corresponding Security Considerations. Jens-Peter Redlich, Wolf Müller. 10 pages.
3. SAR-PR-2005-03: Hacking the Netgear wgt634u. Jens-Peter Redlich, Anatolij Zubow, Wolf Müller, Mathias Jeschke, Jens Müller. 16 pages.
4. SAR-PR-2005-04: Sicherheit in selbstorganisierenden drahtlosen Netzen. Ein Überblick über typische Fragestellungen und Lösungsansätze. Torsten Dänicke. 48 Seiten.
5. SAR-PR-2005-05: Multi Channel Opportunistic Routing in Multi-Hop Wireless Networks using a Single Transceiver. Jens-Peter Redlich, Anatolij Zubow, Jens Müller. 13 pages.
6. SAR-PR-2005-06, NLE-PR-2005-81: Access Control for off-line Beamer – An Example for Secure PAN and FMC. Jens-Peter Redlich, Wolf Müller. 18 pages.
7. SAR-PR-2005-07: Software Distribution Platform for Ad-Hoc Wireless Mesh Networks. Jens-Peter Redlich, Bernhard Wiedemann. 10 pages.
8. SAR-PR-2005-08, NLE-PR-2005-106: Access Control for off-line Beamer Demo Description. Jens Peter Redlich, Wolf Müller, Henryk Plötz, Martin Stigge. 18 pages.
9. SAR-PR-2006-01: Development of a Software Distribution Platform for the Berlin Roof Net (Diplomarbeit / Masters Thesis). Bernhard Wiedemann. 73 pages.
10. SAR-PR-2006-02: Multi-Channel Link-level Measurements in 802.11 Mesh Networks. Mathias Kurth, Anatolij Zubow, Jens Peter Redlich. 15 pages.
11. SAR-PR-2006-03, NLE-PR-2006-22: Architecture Proposal for Anonymous Reputation Management for File Sharing (ARM4FS). Jens Peter Redlich, Wolf Müller, Henryk Plötz, Martin Stigge, Torsten Dänicke. 20 pages.
12. SAR-PR-2006-04: Self-Replication in J2me Midlets. Henryk Plötz, Martin Stigge, Wolf Müller, Jens-Peter Redlich. 13 pages.
13. SAR-PR-2006-05: Reversing CRC – Theory and Practice. Martin Stigge, Henryk Plötz, Wolf Müller, Jens-Peter Redlich. 24 pages.
14. SAR-PR-2006-06: Heat Waves, Urban Climate and Human Health. W. Endlicher, G. Jendritzky, J. Fischer, J.-P. Redlich. In: Kraas, F., Th. Krafft & Wang Wuyi (Eds.): Global Change, Urbanisation and Health. Beijing, Chinese Meteorological Press.
15. SAR-PR-2006-07: 无线传感器网络研究新进展 (State of the Art in Wireless Sensor Networks). 李刚 (Li Gang), 伊恩斯•彼得•瑞德里希 (Jens Peter Redlich)
16. SAR-PR-2006-08, NLE-PR-2006-58: Detailed Design: Anonymous Reputation Management for File Sharing (ARM4FS). Jens-Peter Redlich, Wolf Müller, Henryk Plötz, Martin Stigge, Christian Carstensen, Torsten Dänicke. 16 pages.
17. SAR-PR-2006-09, NLE-SR-2006-66: Mobile Social Networking Services Market Trends and Technologies. Anett Schülke, Miquel Martin, Jens-Peter Redlich, Wolf Müller. 37 pages.
18. SAR-PR-2006-10: Self-Organization in Community Mesh Networks: The Berlin RoofNet. Robert Sombrutzki, Anatolij Zubow, Mathias Kurth, Jens-Peter Redlich, 11 pages.

19. SAR-PR-2006-11: Multi-Channel Opportunistic Routing in Multi-Hop Wireless Networks. Anatolij Zubow, Mathias Kurth, Jens-Peter Redlich, 20 pages.
20. SAR-PR-2006-12, NLE-PR-2006-95: Demonstration: Anonymous Reputation Management for File Sharing (ARM4FS). Jens-Peter Redlich, Wolf Müller, Henryk Plötz, Christian Carstensen, Torsten Dänicke. 23 pages.
21. SAR-PR-2006-13, NLE-PR-2006-140: Building Blocks for Mobile Social Networks Services. Jens-Peter Redlich, Wolf Müller. 25 pages.
22. SAR-PR-2006-14: Interrupt-Behandlungskonzepte für die HHI CineCard-Familie. Robert Sperling. 83 Seiten.
23. SAR-PR-2007-01: Multi-Channel Opportunistic Routing. Anatolij Zubow, Mathias Kurth, Jens-Peter Redlich, 10 pages. IEEE European Wireless Conference, Paris, April 2007.
24. SAR-PR-2007-02: ARM4FS: Anonymous Reputation Management for File Sharing. Jens-Peter Redlich, Wolf Müller, Henryk Plötz, Christian Carstensen, 10 15 pages.
25. SAR-PR-2007-03: DistSim: Eine verteilte Umgebung zur Durchführung von parametrisierten Simulationen. Ulf Hermann. 26 Seiten.
26. SAR-SR-2007-04: Architecture for applying ARM in optimized pre-caching for Recommendation Services. Jens-Peter Redlich, Wolf Müller, Henryk Plötz, Christian Carstensen. 29 pages.
27. SAR-PR-2007-05: Auswahl von Internet-Gateways und VLANs im Berlin RoofNet. Jens Müller. 35 Seiten.
28. SAR-PR-2007-06: Softwareentwicklung für drahtlose Maschennetzwerke – Fallbeispiel: BerlinRoofNet. Mathias Jeschke, 48 Seiten.
29. SAR-SR-2007-07, NLE-SR-2007-88: Project Report: Anonymous Attestation of Unique Service Subscription (AAUSS). Jens-Peter Redlich, Wolf Müller, 19 pages.
30. SAR-PR-2007-08: An Opportunistic Cross-Layer Protocol for Multi-Channel Wireless Networks. Anatolij Zubow, Mathias Kurth, Jens-Peter Redlich, 5 pages. 18th IEEE PIMRC, Athens, Greece, 2007.
31. SAR-PR-2007-09: 100% Certified Organic: Design and Implementation of Self-Sustaining Cellular Networks. Nathanael A. Thompson, Petros Zerfos, Robert Sombrutzki, Jens-Peter Redlich, Haiyun Luo. ACM HotMobile'08. Napa Valley (CA), United States, Feb 25-26, 2008.
32. SAR-SR-2007-10, NLE-SR-2007-88: Project Report: Summary of encountered Security / Performance / Scalability problems with uPB and Wireless Thin Client Architecture. Jens-Peter Redlich, Wolf Müller, 26 pages.
33. SAR-PR-2008-01:On the Challenges for the Maximization of Radio Resources Usage in WiMAX Networks. Xavier Perez-Costa*, Paolo Favaro*, Anatolij Zubow, Daniel Camps* and Julio Arauz*, Invited paper to appear on 2nd IEEE Broadband Wireless Access Workshop colocated with IEEE CCNC 2008. *NEC Laboratories Europe, Network Research Division, Heidelberg, Germany
34. SAR-PR-2008-02: Cooperative Opportunistic Routing using Transmit Diversity in Wireless Mesh Networks. Mathias Kurth, Anatolij Zubow, Jens-Peter Redlich. 27th IEEE INFOCOM, Phoenix, AZ, USA, 2008.

35. SAR-PR-2008-03: Evaluation von Caching-Strategien beim Einsatz von DHTs in drahtlosen Multi-Hop Relay-Netzen - Am Beispiel eines verteilten Dateisystems. Felix Bechstein. Studienarbeit.
36. SAR-PR-2008-04: Precaching auf mobilen Geräten. Sebastian Ehrich, Studienarbeit, 23 Seiten.
37. SAR-PR-2008-05: Towards using 900 MHz for Wireless IEEE 802.11 LANs - Measurements in an Indoor Testbed. Matthias Naber, Moritz Grauel, Studienarbeit, 58 Seiten.
38. SAR-SR-2008-06, NLE-SR-2007-88: Project Report: Standardization / Competitor / Free Software activities related to uPB and Wireless Thin Client – An Overview. Jens-Peter Redlich, Wolf Müller, 13 pages.
39. SAR-PR-2008-07: XtSpaces - A language binding for XVSM. Diploma thesis. Christian Föllmer, 101 Pages.