Network Architectures
and Services
NET 2012-04-1

FI & IITM
WS 11/12

Proceedings of the Seminars
Future Internet (FI) and
Innovative Internet Technologies and Mobile Communications (IITM)

Winter Semester 11/12

Munich, Germany, 13.10.2011-10.02.2012

Editors

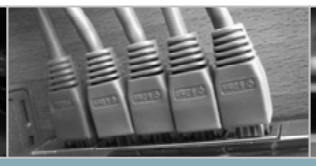
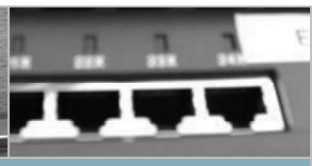
Georg Carle, Corinna Schmitt

Organisation

Chair for Network Architectures and Services
Department of Computer Science, Technische Universität München

Technische Universität München



Network Architectures
and Services
NET 2012-04-1

FI & IITM
WS 11/12

Proceedings zu den Seminaren
Future Internet (FI) und
Innovative Internettechnologien und Mobilkommunikation (IITM)

Wintersemester 2011/2012

München, 13.10.2011 – 10.02.2012

Editoren: Georg Carle, Corinna Schmitt

Organisiert durch den Lehrstuhl Netzarchitekturen und Netzdienste (I8),
Fakultät für Informatik, Technische Universität München

Technische Universität München

Proceedings of the Seminars
Future Internet (FI) and Innovative Internet Technologies and Mobile Communications (IITM)
Winter Semester 2011/2012

Editors:

Georg Carle
Lehrstuhl Netzarchitekturen und Netzdienste (I8)
Technische Universität München
D-85748 Garching b. München, Germany
E-mail: carle@net.in.tum.de
Internet: http://www.net.in.tum.de/~carle/

Corinna Schmitt
Lehrstuhl Netzarchitekturen und Netzdienste (I8)
Technische Universität München
D-85748 Garching b. München, Germany
E-mail: schmitt@net.in.tum.de
Internet: http://www.net.in.tum.de/~schmitt/

Cataloging-in-Publication Data

Seminar FI & IITM WS 2011/2012
Proceedings zu den Seminaren „Future Internet“ (FI) und „Innovative Internettechnologien und Mobilkommunikation“ (IITM)
München, Germany, 13.10.2011-10.02.2012
Georg Carle, Corinna Schmitt
ISBN3-937201-26-2

ISSN: 1868-2634 (print)
ISSN: 1868-2642 (electronic)
DOI: 10.2313/NET-2012-04-2
Lehrstuhl Netzarchitekturen und Netzdienste (I8) NET 2012-04-2
Series Editor: Georg Carle, Technische Universität München, Germany

Vorwort

Wir präsentieren Ihnen hiermit die Proceedings zu den Seminaren „Future Internet“ (FI) und “Innovative Internettechnologien und Mobilkommunikation” (IITM), die im Wintersemester 2011/2012 an der Fakultät Informatik der Technischen Universität München stattfanden.
Im Seminar FI wurden Beiträge zu unterschiedlichen Fragestellungen aus den Gebieten Internettechnologien und Mobilkommunikation vorgestellt. Die folgenden Themenbereiche wurden abgedeckt:

- Verständnis von Schlüssel-unabhängiger Kryptoanalysetechniken zum Angriff auf RC4 basierend auf „Office Encryption stream cipher“
- Niedrige Wahrscheinlichkeit, hohe Einsätze: Eine Betrachtung von PKI
- Hoch performante Client Server Infrastrukturen

Im Seminar IITM wurden Vorträge zu verschiedenen Themen im Forschungsbereich Sensorknoten vorgestellt. Die folgenden Themenbereiche wurden abgedeckt:

- Datenschutz und Smart Meters / Smart Grid
- Einführung in Netzwerk-Fluss-Problem: Das min-cut max-flow Theorem
- TETRA und seine Anwendung bei Behörden und Organisationen mit Sicherheitsaufgaben
- WebSockets: Spezifikation / Implementierung
- Minimum-Cost Multicast über “Coded Packet Networks”

Wir hoffen, dass Sie den Beiträgen dieser Seminare wertvolle Anregungen entnehmen können. Falls Sie weiteres Interesse an unseren Arbeiten habe, so finden Sie weitere Informationen auf unserer Homepage http://www.net.in.tum.de.

München, April 2012

Georg Carle

Corinna Schmitt

Preface

We are very pleased to present you the interesting program of our main seminars on “Future Internet” (FI) and “Innovative Internet Technologies and Mobil Communication” (IITM) which took place in the winter semester 2011/2012

In the seminar FI we deal with issues of Future Internet. The seminar language was German, and the majority of the seminar papers are also in German. The following topics are covered by this seminar:

- Understanding the key-independent cryptanalysis technique used to break the RC4 based Office Encryption stream cipher
- Low Probability, High Stakes: A look at PKI
- High Performance Client Server Infrastructure

In the seminar IITM talks to different topics in innovate internet technologies and mobile communications were presented. The seminar language was German, and also the seminar papers. The following topics are covered by this seminar:

- Privacy and Smart Meters / Smart Grid
- Introduction to Network Flow Problems: The min-cut max-flow Theorem
- TETRA and its Applications in Administration and Organization Facilities dealing with Security Tasks
- WebSockets: Specification and Implementation
- Minimum-Cost Multicast over Coded Packet Networks

We hope that you appreciate the contributions of these seminars. If you are interested in further information about our work, please visit our homepage http://www.net.in.tum.de.

Munich, April 2012

Seminarveranstalter

Lehrstuhlinhaber

Georg Carle, Technische Universität München, Germany

Seminarleitung

Corinna Schmitt, Technische Universität München, Germany

Betreuer

Philipp Fehre, *Technische Universität München, Wiss. Mitarbeiterin I8*
Stephan Günther, *Technische Universität München, Wiss. Mitarbeiter I8*
Ralph Holz, *Technische Universität München, Wiss. Mitarbeiter I8*
Heiko Niedermayer, Technische Universität München, Wiss. Mitarbeiter I8

Kontakt:

{carle,schmitt,fehre,guenther,holz,niedermayer }@net.in.tum.de

Seminarhomepage

http://www.net.in.tum.de/de/lehre/ws1112/seminare/

Inhaltsverzeichnis

Seminar Future Internet

Seminar Innovative Internettechnologien und Mobilkommunikation

Understanding the key-independent cryptanalysis technique used to break the RC4 based Office Encryption stream cipher

Elias Tatros
Betreuer: Heiko Niedermayer
Seminar Future Internet WS10/11
Lehrstuhl Netzarchitekturen und Netzdienste
Fakultät für Informatik, Technische Universität München
Email: elias.tatros@cs.tum.edu

ABSTRACT

Stream ciphers are still widely used to efficiently encrypt large amounts of data. Contrary to asymmetric encryption, stream ciphers are easy to implement in hardware and offer a high encryption speed. Using information theory, it is possible to proof that an arbitrary level of security can be achieved. However small flaws in the implementation of stream ciphers can lead to severe weaknesses. These can cause even unconditionally secure stream ciphers, like the one-time pad (Vernam system), to become totally insecure. This can be demonstrated in a cryptanalysis of the Microsoft Office encryption up to the 2003 version. Office employs a flawed implementation of the widely used RC4 algorithm, exposing a severe vulnerability, which can be exploited to recover the plaintext from the encrypted documents.

Keywords

stream ciphers, data encryption, code breaking, applied cryptography, RC4

1. INTRODUCTION

Many cryptographic solutions, that are considered secure in a theoretical sense, do not necessarily offer the same security when implemented in a real cryptosystem. It is fairly common that the key size alone serves as the most important security argument. A cryptosystem based on AES-256 might not be secure if its implementation is flawed. It is most probably not secure, if intended implementation backdoors were deliberately introduced into the system. Thus, in practice, an argument like "AES-256 inside" can only be considered a necessary, but not a sufficient condition for a secure solution.

A different problem, that can have a dramatic impact on the security, lies in the misuse of a cryptosystem. When users ignore basic rules for handling the system, then even information theoretically secure cryptosystems can become totally insecure. One prominent example for such a misuse occurred in the 1940s, when US intelligence under the VENONA project was able to break the one-time pad encryption on many high-level soviet diplomatic messages. The successful cryptanalysis of many of those encrypted messages was possible, because the people handling the encryption process for the soviets made mistakes using the cryptosystem. They reused some of the secret keys (called pads, because at that time they were printed on pads of paper) on different messages, thus ignoring a basic rule for handling one-time pad systems stating that a secret key must never be used more than once. When a cryptosystem is labeled (information theoretically) secure its users tend to have high confidence in the system and may therefore be less likely to thoroughly question their own actions when handling it.

From these issues two problems immediately arise. For users of a cryptosystem the problem is to detect implementation weaknesses or trapdoors within the system without reverse engineering. Reverse engineering is not a viable solution to the problem because in general it is too time consuming and quite often also illegal. For an attacker the problem is to identify, among a large number of ciphertexts, those that were generated by a weak or misused cryptosystem and then to recover the corresponding plaintexts in a reasonable amount of time. This needs to be done without knowledge of the cryptographic algorithm. For example, in satellite transmissions it can be assumed that the attacker is able to intercept the ciphertexts but doesn't know the algorithm used for encryption.

2. SYMMETRIC ENCRYPTION BASICS

Whenever the confidentiality of data is of concern encryption must be used. For large amounts of data it is better to use symmetric encryption. Advantages over asymmetric encryption include faster encryption speed and in most cases a high level of error resilience, while still offering an arbitrary level of security. Symmetric encryption solutions can be categorized into stream ciphers and block ciphers.

When using stream ciphers bits (or bytes) are enciphered and deciphered on-the-fly. This enables the cryptosystem to rapidly encrypt large quantities of data. They are also very error resilient since transmission errors do not propagate during the decipherment. For these reasons they are widely used for satellite communications protection, telephony encryption (e.g. A5/1) and Bluetooth encryption (e.g. E0) [1].

Contrary to stream ciphers, bits are not enciphered or deciphered on-the-fly when using block ciphers. Instead data is first split into blocks of a certain size (the standard is 128-bit blocks). Each of these blocks is then enciphered or

doi: 10.2313/NET-2012-04-1_1

deciphered using the same secret key. Block ciphers can operate in a number of modes which can provide authentication, enable different encryption properties and limit error propagation. The most common modes are ECB, CBC, CFB, PCBC, OFB [2]. It is worth noting that block ciphers, except in OFB mode, are not naturally transmission error resilient. Block ciphers in output feedback mode (OFB) emulate stream ciphers and thus are susceptible to stream cipher attacks and misuses (e.g. key reuse) [1].

2.1 Stream Ciphers in Detail

A stream cipher bitwise combines a truly random or pseudorandom sequence $(\sigma_t)_{t\geq 0}$ with the plaintext $(m_t)_{t\geq 0}$ using the xor operation which results in the ciphertext $(c_t)_{t\geq 0}$:

$$c_t = m_t \oplus \sigma_t$$

Since the xor operation is involutive, it is sufficient to apply the same random bit σ_t to ciphertext bit c_t for the time instant t and thus recover the deciphered plaintext bit m_t:

$$m_t = c_t \oplus \sigma_t$$

where c_t, m_t and σ_t denote a ciphertext, plaintext and random sequence bit at time instant t respectively. The sequence $(\sigma_t)_{t\geq 0}$ is called the running-key. In the case of Vernam ciphers (one-time pad) the running-key is truly random and independent from plaintext or ciphertext (i.e. produced by hardware methods). The one-time pad encryption has been proven to be information theoretically secure, meaning that the ciphertext C provides no information about the plaintext M to a cryptanalyst.

2.2 Perfect Secrecy

Perfect secrecy, as defined in [3], means that after the interception of a ciphertext the *a posteriori* probabilities of this ciphertext representing various plaintexts be identical to the *a priori* probabilities of the same plaintext before the interception, implying that intercepting ciphertexts doesn't provide the cryptanalyst with any information about the plaintexts. It is possible to achieve perfect secrecy, but it requires that there is at least one key k transforming any plaintext M into any ciphertext C. Obviously any key from a fixed M to a different C must be different (as shown in figure 1), therefore the number of unique keys must be at least as high as the number of plaintexts.

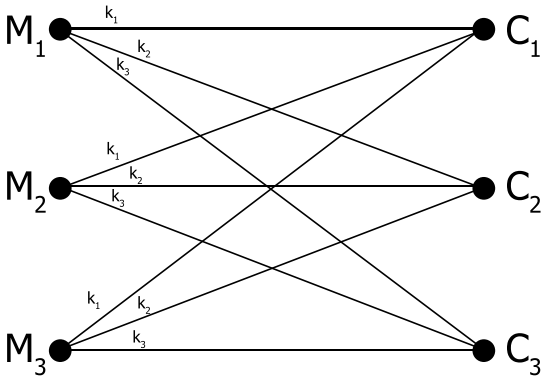


Figure 1: a perfect system as described in [3]

More precisely, if a finite key was used for encryption and a ciphertext consisting of N letters is intercepted, there will be a set of possible plaintexts with certain probabilities that this ciphertext could represent. C. E. Shannon has shown in [3] that as N increases the probabilities for all possible plaintexts except for one will approach zero. It is now possible to define a quantity $H(N)$ that measures by statistical means how near the average ciphertext of N letters is to the unique solution (i.e. how uncertain a cryptanalyst is of the plaintext after intercepting N letters of ciphertext). As shown in [4], this quantity can be measured by means of entropy. For a given set of possibilities with probabilities $p_1, p_2, ..., p_n$ the entropy H is given by:

$$H = -\sum p_i \log p_i$$

The perfect secrecy property can then be expressed as:

$$H(M) = H(M|C)$$

Intuitively this means that the uncertainty about the plaintext remains constant with each interception of a new ciphertext. The same holds true when applying the entropy function to the key:

$$H(K) = H(K|C)$$

As previously mentioned, it is necessary to have

$$H(K) \geq H(M)$$

in order to achieve perfect secrecy. If the key bits are truly randomly produced (i.e. by a true random number generator) then $H(K) = |K|$, meaning that the running-key $(\sigma_t)_{t\geq 0}$ must be at least as long as the message to encipher. From a practical point of view, this property becomes very difficult to handle for large-scale operations (e.g. soviets reused keys for one-time pad encryption and thus enabled cryptanalysis in VENONA project) because the hardware based generation of truly random sequences is somewhat complex and doesn't scale very well. This is the reason why one-time pads are generally used for short messages or strategic use only. Usually in stream ciphers the running-key is produced by a PRNG (pseudorandom number generator) as an expansion of a secret key that is reduced in size (up to 256-bits). The secret key only serves for the initialization of the algorithm at time $t = 0$. But now $H(K) \geq H(M)$ is no longer true. Yet, as stated in [1], in practice the number of messages enciphered by a cryptosystem using |K|-bit keys is always far lower than $2^{|K|}$ (the number of secret keys), thus it can be assumed that during the lifetime of the cryptosystem

$$H(K) \geq H(M)$$

holds true. However, like in one time pad systems, it is very important not to reuse secret keys. The security of stream ciphers becomes void whenever a secret key is reused to initialize the system.

2.3 Describing the Attack

The preceding chapters served as a basic introduction to symmetric encryption. In the following chapters the attack

doi: 10.2313/NET-2012-04-1_1

itself will be explained. At first in a general theoretical way applicable to any stream cipher and then specifically applied to the Microsoft Office 2003 RC4 encryption. Chapters 3 and 4 explain how to detect ciphertexts in which the key has been reused. These chapters correspond to sections 6.2 and 6.4 which highlight the vulnerability in the MS Word case. Here the IV is reused in revised versions of Word documents which leads to parallel ciphertexts that can be obtained from temporary files in Windows. Chapter 5 describes a general algorithm to recover the plaintexts from the ciphertexts by statistical means. The cryptanalysis consists of a language specific frequency analysis of character sequences. This chapter corresponds to section 6.5, which shows and evaluates the results obtained in tests performed on french texts in MS Word documents.

3. PARALLEL CIPHERTEXTS

As shown by C.E. Shannon in [4] the security of stream ciphers and block ciphers in OFB mode is nullified if the secret key is reused. In order to clarify why key reuse compromises the security of any such cipher, it is necessary to take a look at what exactly happens when the same secret key is used to encipher multiple plaintexts. Let m_1 and m_2 be plaintexts and let σ be the single secret key used to encipher both of them into the resulting ciphertexts c_1 and c_2:

$$c_1 = m_1 \oplus \sigma$$
$$c_2 = m_2 \oplus \sigma$$

Then the ciphertexts c_1 and c_2 are said to be *parallel*.

More generally, as defined in [1], two (or more) ciphertexts are said to be parallel if they are produced either by a stream cipher (Vernam cipher or finite state machine) or by a block cipher in OFB mode using the same running-key. Furthermore the parallelism depth of k parallel ciphertexts $c_1, c_2, ..., c_k$ is k. Whenever a key is reused and parallel ciphertexts occur, the perfect secrecy as defined by Shannon in [3] has been violated and the parallel ciphertexts should be susceptible to cryptanalysis.

The first step necessary in order to perform a successful cryptanalysis is to detect groups of these parallel ciphertexts among a large number of ciphertexts. Detection needs to work without knowledge of the underlying cryptosystem (other than that it is indeed a stream cipher or block cipher in OFB mode), meaning the algorithm used for encryption can remain unknown. This is important because, as covered in the introduction, when ciphertexts have been intercepted the crypotsystem is usually still unknown and most of the time it is not feasible to uncover the algorithm using reverse engineering. It is also very interesting to note that whenever parallel ciphertexts will be detected, one can assume that either a serious misuse has occurred or that an implementation flaw, or worse, trapdoor exists within the program.

Once parallel ciphertexts have been detected, the logical follow up step is to perform a cryptanalysis to break the encryption and recover the plaintexts. This needs to be done using the ciphertexts alone, meaning the underlying cryptosystem and the key used for encryption may remain unknown. Since there is no preliminary key recovery, this technique is described in [1] as a *key-independent cryptanalysis*.

4. DETECTING PARALLEL CIPHERTEXTS

As explained in the previous chapter, parallel ciphertexts come into existence when multiple plaintexts are encrypted with the same secret key. They can then be exploited to break the encryption and recover the plaintexts without knowledge of the key or algorithm used. This chapter explains how to detect groups of parallel ciphertexts (if any) among a large number of ciphertexts.

4.1 Statistical Features in Languages

In order to understand the following steps it is important to know that plaintexts exhibit strong statistical features which depend on the language and encoding of the text. As explained by C.E. Shannon in [3] a language can be described as a stochastic process that produces a certain sequence of symbols according to a system of probabilities. He defines the parameter D as the redundancy of the language. Intuitively speaking, D measures how much text in a language can be reduced without losing any information. For example, in the English language the letter u may be omitted without loss of information when occuring after the letter q. This is possible because u always follows the letter q, thus it is sufficient to keep the q and discard u. Due to the statistical structure of the English language (i.e. high frequency of certain letters or words) many such reductions are possible. In fact, given a certain language with parameter D it is possible to calculate the number of intercepted ciphertext letters required to obtain a statistical solution:

$$\frac{H(K)}{D}$$

where H(K) is the size of the key space, which for n possible messages that are all a priori equally likely cannot exceed $\log_2 n$ bits. For example, in a very simple monoalphabetic substitution of letters $H(K) = \log_2(26!) \approx 88.3$ and $D \approx 3.2$ for the English language, hence about 28 letters are sufficient to break the encryption.

Furthermore it is interesting to note that encodings can hide or amplify certain statistical features and thus the choice of encoding is of importance when considering a trapdoor design. The next step is to build a statistical hypothesis test that serves as our general detection method and determines whether two ciphertexts are parallel or not.

4.2 Forming the Statistical Hypotheses

Consider the stream cipher xor encryption and let all operations be bitwise or bytewise as usual. Let

$$C_1 = M_1 \oplus \sigma_1$$
$$C_2 = M_2 \oplus \sigma_2$$

where M_1, M_2 are plaintexts, σ_1, σ_2 are the keys and C_1, C_2 are the resulting ciphertexts. From the previous chapter it is known that plaintexts M_1 and M_2 exhibit very strong statistical features depending on language and encoding (i.e. each character - letter, number, punctuation - has a different frequence of occurrence). Considering the ciphertexts C_1 and C_2 this frequence of occurrence is different. When using

doi: 10.2313/NET-2012-04-1_1

ASCII encoding the probability for each character is $\frac{1}{256}$. Therefore the quantity

$$M_1 \oplus M_2$$

exhibits a very special statistical profile that can be detected and identified, whereas the quantity

$$\sigma_1 \oplus \sigma_2$$

exhibits a totally random statistical profile. The xor of the two ciphertexts C_1 and C_2 gives:

$$C_1 \oplus C_2 = M_1 \oplus \sigma_1 \oplus M_2 \oplus \sigma_2$$

If $\sigma_1 \neq \sigma_2$ the quantity $C_1 \oplus C_2$ will exhibit a totally random profile. But if the secret key has been reused (i.e. C_1 and C_2 are parallel and $\sigma_1 = \sigma_2$) then:

$$\begin{aligned} C_1 \oplus C_2 &= M_1 \oplus \sigma_1 \oplus M_2 \oplus \sigma_2 \\ &= M_1 \oplus M_2 \end{aligned}$$

Therefore the quantity $C_1 \oplus C_2$ exhibits a very strong statistical profile whenever $\sigma_1 = \sigma_2$ and a totally random statistical profile otherwise. Since this quantity behaves differently whenever key reuse occurs, it can be used to form statistical hypotheses in the statistical test for detection of parallel ciphertexts. The two hypotheses can now be defined as:

- Null hypothesis (H_0): $\sigma_1 \neq \sigma_2$. Ciphertexts C_1 and C_2 are not parallel, since the key has not been reused. Therefore the quantity $C_1 \oplus C_2$ exhibits a random statistical profile.
- Alternative hypothesis (H_1): $\sigma_1 = \sigma_2$. Ciphertexts C_1 and C_2 are parallel, the key has been reused. Thus the quantity $C_1 \oplus C_2$ is exactly equal to $M_1 \oplus M_2$ and therefore exhibits the same special statistical profile.

4.3 Choosing a Suitable Estimator

With the two hypotheses, established in the previous chapter, it is now possible to build an estimator that behaves differently in the cases (H_0) and (H_1) and therefore is suitable to detect whether two ciphertexts are parallel or not. Detecting parallel ciphertexts among a large number of ciphertexts is now as easy as detecting non-random files from random files. The chosen estimator in [1] performs a bitwise xor operation on each pair of ciphertexts and then counts the number of bits equal to null in the resulting sequence. Let n be the common part of the length of the two ciphertexts C_1 and C_2 in bits, while c_1^i and c_2^i represent ciphertext bits of C_1 and C_2 at time instant i respectively, then the number of nullbits in the xor of C_1 and C_2 is denoted as Z:

$$Z = \sum_{i=0}^{n} (c_1^i \oplus c_2^i \oplus 1)$$

Now let p be the probability that a bit in the sequence $C_1 \oplus C_2$ is equal to zero:

$$p = P[c_1^i \oplus c_2^i = 0]$$

Since every bit in the sequence $C_1 \oplus C_2$ is either null or one with a certain probability, p or $(1-p)$ respectively, and each xor result is independent from the previous one, Z has a binomial distribution with parameter n and p. For large n the binomial distribution can then, by application of the *de Moivre-Laplace theorem* [5], be approximated by a normal distribution with mean value np and standard deviation $\sqrt{np(1-p)}$. Therefore, assuming large enough n, Z has normal distribution:

$$Z \sim \mathcal{N}(np, \sqrt{np(1-p)})$$

Using this result it is possible to detect parallel ciphertexts from non-parallel ciphertexts, since the probability p is different with respect to hypotheses H_0 and H_1. For non-parallel ciphertexts, because of their random statistical profile, $p = \frac{1}{2}$. For parallel ciphertexts p depends on language and encoding. As stated by E. Filiol in [1], one can assume $p > 0.6$ for most languages. Therefore the setup for the statistical hypotheses test is now complete:

- If $Z \sim \mathcal{N}(\frac{n}{2}, \frac{\sqrt{n}}{2})$ then assume H_0: The key has not been reused ($\sigma_1 \neq \sigma_2$), consequently the ciphertexts C_1 and C_2 are not parallel.
- If $Z \sim \mathcal{N}(np, \sqrt{np(1-p)})$ with $p > \frac{1}{2}$ then assume H_1: The key has been reused ($\sigma_1 = \sigma_2$), consequently the ciphertexts C_1 and C_2 are parallel.

In all experiments carried out by E. Filiol in [1], large peak values for Z (usually above 0.6) were observed whenever ciphertexts were parallel. Therefore this test has also been empirically verified.

4.4 Detection Algorithm and Error Reduction

To find and further reduce any errors during detection, it is possible to apply an equivalence relation over the set of parallel ciphertexts. Let C_i, C_j and C_k be any ciphertexts and $\mathcal{R}$ the relationship "be parallel to". Then $\mathcal{R}$ is an equivalence relation over the set of parallel ciphertexts, i.e. it is:

- reflexive: $C_i \mathcal{R} C_i$, since obviously any ciphertext is parallel to itself.
- symmetric: $C_i \mathcal{R} C_j \rightarrow C_j \mathcal{R} C_i$, meaning the direction of the parallel relation is not relevant.
- transitive: $C_i \mathcal{R} C_j \wedge C_j \mathcal{R} C_k \rightarrow C_i \mathcal{R} C_k$, which can be used as a consistency check in the detection process.

The equivalence relation $\mathcal{R}$ partitions the set of parallel ciphertexts into equivalence classes. Any such class forms a group of parallel ciphertexts. This can be used for consistency checks, since any ciphertext in a certain equivalence class can never be in a different equivalence class at the same time. Furthermore, using the transitivity property of $\mathcal{R}$, if C_1 is detected to be parallel with C_2 and C_2 is parallel with C_3, then C_1 must also be parallel to C_3 and all three ciphertexts must be exclusively in the same equivalence class. Any violation of these rules would point to an error in the detection process and require further decision making by either the algorithm or the user.

The general detection algorithm only requires a set of ciphertexts as the input. It then simply compares all ciphertexts pairwise, using the statistical hypotheses test. Finally it builds groups of parallel ciphertexts using the $\mathcal{R}$ relation to check for consistency.

doi: 10.2313/NET-2012-04-1_1

Letter	Frequency	Letter	Frequency
A	6.09	N	5.44
B	1.05	O	6.00
C	2.84	P	1.95
D	2.29	Q	0.24
E	11.36	R	4.95
F	1.79	S	5.68
G	1.38	T	8.03
H	3.41	U	2.43
I	5.44	V	0.97
J	0.24	W	1.38
K	0.41	X	0.24
L	2.92	Y	1.30
M	2.76	Z	0.03

Table 1: relative frequency of English letters

5. RECOVERING THE PLAINTEXTS

The question, how to detect parallel ciphertexts, may they be caused by key misuse, implementation flaws or intentional trapdoors, has been taken care of in the previous chapter. The next logical step is to proceed with a cryptanalysis in an effort to recover the plaintexts by exploitation of the detected flaws. As mentioned in chapter three and four, the cryptanalysis technique used by Filiol in [1] will recover the plaintexts without the key. In fact the algorithm and key used for encryption are irrelevant. It is however very important to generate a reliable and conclusive statistical model of the target language. The term language is considered in the general, formal sense and not limited to natural languages. Consequently this approach works for all languages generated by any of the four grammar types in the Chomsky hierarchy. Furthermore, when building the model, it is also important to consider the encoding used (ASCII, Unicode, ...).

5.1 Constructing the Corpus

The concept of a corpus is defined in [1] as the set of all possible n-grams with their respective frequency of occurrence in the target language. An n-gram is simply understood as a string of n characters of that language. The corpus will serve as a qualitative and quantitative model of the target language. Any language can be described by the frequency of occurrence of its characters. One can easily select a few relevant texts and build a table with single letters (or other characters) and their frequency of occurrence in those texts. As an example, table 1 shows the english letter frequency taken from a large text, as determined in [6].

The same can be done for all possible n-grams and their respective frequencies in order to build the final corpus. Filiol has shown in [1], that 4-grams are the best choice when considering memory, time and accuracy of the model.

The quantitative aspect obviously lies in its size N, which is given by the number of n-grams and in their assigned frequencies. But there is also an important qualitative aspect of the corpus, which must be considered carefully. The corpus must be representative of the target language. In order to build the corpus a set of text is searched and frequencies of the n-grams are extracted. The choice of those texts must be made wisely. There's a large number of texts from

a	b	c	d	e	f	g	h	i	j
k	l	m	n	o	p	q	r	s	t
u	v	w	x	y	z	A	B	C	D
E	F	G	H	I	J	K	L	M	N
O	P	Q	R	S	T	U	V	W	X
Y	Z	0	1	2	3	4	5	6	7
8	9	.	,	;	:	?	!	≪	(
)	{	}	+	-	*	/	=	'	à
â	é	ç	è	é	ê	î	ô	ù	

Table 2: French language character space

many different times using different levels of language (i.e. common, technical, political, diplomatic, military, ...) that must be considered. It is therefore a good idea to build a number of different corpora, which model these different levels of language very precisely and then choose whatever corpus is best suited for the cryptanalysis, depending on the operational context at hand. A further requirement is that the texts contain a statistically significant amount of characters. It is interesting to note, that the corpus of n-grams is generally compliant to Zipf's law when considering natural languages [7]. This means, that the frequency of any n-gram is inversely proportional to its rank in the corpus.

In order to limit the resources required for the corpus, it is important to limit the character space as much as possible without omitting critical characters needed to describe the language effectively. This step is especially crucial for languages that contain a large amount of characters (e.g. due to their accentuation). This is true for languages modeled in ASCII encoding, such as French, Turkish and several northern european languages. When considering Asian or Arabic languages the same approach can be used, but a different encoding must be considered. Table 2 shows the character space chosen by Filiol in [1] for the French language.

Another important criterion is the length of the n-grams, i.e. the choice of n. This choice directly influences the size of the corpus. Obviously for larger n more combinations of characters will be possible, thus significantly increasing size and memory requirements of the corpus while also increasing search times. As experimentally verified by Filiol in [1], the best choice for n is $n = 4$, since using tetragrams over trigrams ($n = 3$) greatly improved results but using pentagrams ($n = 5$) did not further improve results. Considering a character space with 95 characters (ASCII) will produce a corpus of 95^4 different tetragrams.

5.2 Algorithm for Plaintext Recovery

Assuming that a number of ciphertexts were intercepted of which p are detected to be parallel, let $C_1, C_2, ..., C_p$ be the p parallel ciphertexts and $M_1, M_2, ..., M_p$ be the corresponding plaintexts. Then using a corpus of N n-grams and the algorithm given by Filiol in [1], it is possible to recover the plaintexts $M_1, M_2, ..., M_p$, without knowledge of the encryption algorithm or key. Before applying the algorithm the p ciphertexts are split into a succession of x n-grams. This algorithm will result in xN p-tuples, meaning N p-tuples for each n-gram at position j in the ciphertexts. Each such p-tuple is of the form $(M_1^j, M_2^j, ..., M_p^j)$ and contains possible plaintext candidates for the n-grams at position j within

doi: 10.2313/NET-2012-04-1_1

the plaintext messages $M_1, M_2, ..., M_p$.

The first step is to make an assumption for the plaintext n-gram M_1^j which corresponds to the ciphertext n-gram C_1^j. This assumption is added to a p-tuple as the first element. Then the key n-gram at position j is given as

$$K_j = C_1^j \oplus M_1^j.$$

In the next step K_j is combined with every C_i^j in the $(p-1)$ remaining ciphertexts, using the xor operation thus generating the remaining $(p-1)$ M_i^j of the p-tuple.

$$M_i^j = C_i^j \oplus K_j, \text{ for } i \in [2, p]$$

These resulting M_i^j represent possible plaintext n-gram solutions at position j for the remaining (p-1) ciphertexts and are added to the p-tuple. These steps are done for every position j, meaning $j \in [1, x]$. Then the complete set of steps is exhaustively repeated N times, meaning each of the N n-grams in the corpus needs to serves as a guess for M_1^j.

After completion the algorithm will have generated xN p-tuples. N p-tuples for each n-gram at position j. Therefore the results are p-tuples of the form $(M_1^j, M_2^j, ..., M_p^j)$, $j \in [1, x]$ being the position of the n-grams within the plaintexts and $i \in [1, p]$ denoting the plaintext. N such p-tuples exist for every position j:

$$\begin{aligned}
j = 1 \quad & (M_1^1, M_2^1, ..., M_p^1),\ (M_1^{'1}, M_2^{'1}, ..., M_p^{'1}),\\
& (M_1^{''1}, M_2^{''1}, ..., M_p^{''1}),\ ...\\
j = 2 \quad & (M_1^2, M_2^2, ..., M_p^2),\ (M_1^{'2}, M_2^{'2}, ..., M_p^{'2}),\\
& (M_1^{''2}, M_2^{''2}, ..., M_p^{''2}),\ ...\\
& \vdots\\
j = x \quad & (M_1^x, M_2^x, ..., M_p^x),\ (M_1^{'x}, M_2^{'x}, ..., M_p^{'x}),\\
& (M_1^{''x}, M_2^{''x}, ..., M_p^{''x}),\ ...
\end{aligned}$$

The next step is to select the most probable of the N p-tuples for each position. In order to do that a p-tuple of probabilities $(P[M_1^j], P[M_2^j], ..., P[M_p^j])$ is associated with each of the corresponding xN p-tuples generated by the algorithm. In order to find the most probable plaintext n-grams p-tuple one has to determine the p-tuple that maximizes the p-tuples of probabilities. For that purpose a suitable function must be chosen that can compute those probabilities in the most significant way:

$$Z_j = f(P[M_1^j], P[M_2^j], ..., P[M_p^j])$$

As explained by Filiol in [1], this step is where the ability and experience of the cryptanalyst becomes important, since the choice of this function strongly depends on the nature and contents of the texts. This function is named the *frequency cumulative function* [1] and must always be strictly increasing and positive. The probability of success depends strongly on the frequency function and a few other parameters (n-gram processing mode and decryption mode) that will be explored later on.

Let f_i be the frequency of occurrence of n-gram i in the corpus, then the most efficient choices for the *frequency cumulative function* are the *additive function* given by

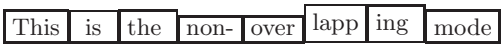
$$\sum_{i=0}^{p} f_i^a$$

and the *multiplicative function* given by

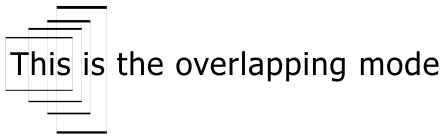
$$\prod_{i=1}^{p} (f_i^a + 1).$$

For texts containing many low frequency words, meaning low values of f_i (e.g. proper names, technical terms), the multiplicative function should be preferred, since it is much more efficient in this case. The optimal value for the parameter a is given by Filiol [1] as $a = 0.3$.

Another important factor for the successful recovery of the plaintexts is the *n-gram processing mode*. This refers to how the ciphertexts are split into n-grams. The first mode and most obvious solution is to simply split the ciphertext into a number of non-overlapping n-grams. This means that two consecutive n-grams have a void intersection. For example, using $n = 4$ this mode would produce the following framed n-grams:



In this mode n-grams do not share any characters which is not optimal if one wants to check for consistency. An advantage is that such a mode is very easy to implement. The disadvantage however is that whenever a wrong plaintext candidate is chosen, this error cannot be detected, since the n-grams are all independent. Therefore a different mode is introduced that splits the texts into n-grams by shifting one character position at a time: The overlapping mode allows



for consistency checks, since two consecutive n-grams always share $(n-1)$ characters. Using this property, one can verify that every n-gram candidate at position $j+1$ has $(n-1)$ common characters with the n-gram candidate at position j. This offers a huge help in selecting the correct plaintext n-gram candidates. However, this mode is a bit more complex to implement than the non-overlapping mode.

So far only the best plaintext n-gram candidate for every ciphertext n-gram at position j was kept (i.e. n-grams from the tuple that maximized the frequency cumulative function). This approach is called hard decoding. A further optimization can be made in keeping the b best candidates (soft decoding). This allows the use of backtracking, thus enabling the correction of wrong decisions.

The last effective optimization presented in [1] is to use the chosen character space as a limiting factor in the plaintext candidate selection. For example, if the plaintexts are known to contain only common language, one can allow only for p-tuples that contain printable characters exclusively to be accepted as candidates. This optimization is easy to implement, speeds up the plaintext recovery in general and also potentially prevents many wrong decisions. Since it is very likely that many p-tuples contain n-gram candidates with non-printable characters this increases the chances of a successful recovery greatly.

doi: 10.2313/NET-2012-04-1_1

Finally, the best approach utilizes all the optimizations presented using parameters that were experimentally verified by Filiol in [1]. This means that the final cryptanalysis algorithm uses a multiplicative frequency function F with parameter $a = 0.3$:

$$F(f_1, f_2, ..., f_p) = \prod_{i=1}^{p} (f_i^a + 1),$$

the overlapping n-gram processing mode with consistency checks and a soft decoding, keeping the b best candidates with $b \in [5, 10]$.

6. BREAKING THE WORD ENCRYPTION

Now all the theoretical concepts needed for detecting a vulnerability (e.g. parallel ciphertexts caused by key misuse, implementation flaw or intentional trapdoor) and the cryptanalysis itself to recover the plaintexts from those parallel ciphertexts without a preliminary key recovery have been established. These concepts can be applied to the Microsoft Word encryption (up to Office version 2003). A number of different encryption methods are offered by Word, the very simple constant XOR encryption, the Office 97/2000 compatible encryption, which is a proprietary Office encryption, derived from the Microsoft Internet Explorer CryptoAPI method and several encryption services that are based on the RC4 stream cipher.

6.1 Word Encryption Methods

The default encryption method used by Office is the constant xor. In this encryption method the plaintext is combined with a constant 16-character string, derived from a user specified password, using the xor operation. The character string is simply repeated to cover all the plaintext. From a security point of view, this encryption method is very weak and in fact there already exist many tools that are dedicated to breaking this type of encryption in minutes (e.g. several products by the company Elcomsoft, which interestingly is also a Microsoft certified partner now). Although it might not be as efficient as dedicated software, it is also possible to utilize the previously discussed key-independent cryptanalysis, developed by Filiol, to break this type of encryption.

However this approach really shines when trying to break the RC4 based encryption methods offered by Word and other Office applications. These consist of several encryption standards (e.g. Diffie-Hellman with DSS and SHA-1), which theoretically offer adequate confidentiality, accountability and integrity. The strongest security is allegedly provided by the *RC4, Microsoft Enhanced Cryptographic Provider* service. *Microsoft Enhanced Cryptographic Prover* provides the same services as the *Microsoft Base Cryptographic Provider* services, but offers additional security through the use of longer keys and additional algorithms. In the RC4 case the key length of the enhanced provider is 128-bits, whereas the base provider only offers a 40-bits key. The *Microsoft Enhanced Cryptographic Prover* encryption (using RC4 with a 128-bit key and SHA-1) is the target of the attack described in [1].

RC4 is a stream cipher (symmetric key algorithm) that was developed by Ronald Rivest in 1987. It is still widely used in many applications that utilize stream ciphers. As usual for stream ciphers, RC4 uses a cryptographic bit stream that is combined with the plaintext using the xor function to produce the ciphertext. In the Office case the cryptographic bit stream is generated by a proprietary algorithm. A secret 128-bit key initializes a so called state table that is used to generate pseudo-random bytes, which are then utilized to generate a pseudo-random bit stream [8]. The 128-bit key is derived by a function F that takes the hash of a user specified password, concatenated with a "randomly" produced initialization vector (IV), to generate 128-bit values. Let F be the key generating function, H a cryptographic hash function (e.g. SHA-1) and IV the initialization vector, then the key K is given by:

$$K = F(H(IV||password))$$

This is a fairly standard approach to generate a key from a password (though usually an iterated hash is recommended) and a first step towards a strong encryption, since the key does not depend on the user's password alone. This is due to the randomly produced IV that is concatenated with the password, thus preventing the reuse of a key, even if the same password is used more than once. As stated by Filiol in [1] the IV plays the same role as a session key.

6.2 The Office Vulnerability

As discussed in chapter two, when using stream ciphers, it is imperative, not to reuse a secret initialization key, ever. One must assume, that many users will use the same password more than once, especially when considering the same document. Since Office generates the secret initialization key by the formula

$$K = F(H(IV||password)),$$

the security of the entire encryption becomes void whenever the initialization vector (IV) is reused. The vulnerability, first identified in [9], now lies in the fact, that Word 2003 (and Office 2003 in general) reuses the same IV for every revised version of a Word document. Since only multiple versions of a single document are considered, one can assume that the user will keep the same password for this document. Altogether this means, that the same secret key will be used for initialization of the RC4 stream cipher, whenever a revised (modified) version of a Word document is saved. As shown by Filiol in [1], this is the case even when a new file name is used to save the modified document. Thus all the revised versions of any single Word document form a set of parallel ciphertexts. Once detected by means of the detection method discussed in chapter four, these parallel ciphertexts can then be subjected to the cryptanalysis described in chapter five and finally the plaintexts can be recovered without any knowledge of the key. The vulnerability is caused by a flawed implementation of the RC4 algorithm in Microsoft Office, not by RC4 itself.

6.3 Word Document Specifics

Before detection of parallel ciphertexts can occur, one must find the encrypted data within the Word documents. As determined by Filiol in [1], the beginning of the encrypted text in any Microsoft Word document is always located at offset $0xA00$. The size S of the encrypted text in bytes is calculated from two values x and y, located at offsets $0x21D$ and $0x21C$. Surprisingly those values are never encrypted,

doi: 10.2313/NET-2012-04-1_1

even if *document properties encryption* is enabled, thus the size in bytes can be calculated using the following formula:

$$S = 256x + y - 2048.$$

6.4 Detecting Parallel Word Documents

In order to apply the key-independent cryptanalysis described in chapter five, it is first necessary to locate several encrypted Word documents and determine whether their contents constitute a parallel group of ciphertexts. The flaw in the implementation of RC4 in Office is further amplified by the way the Windows operating system handles temporary files. Creating or modifying an Office document also creates temporary files, each containing a previous version of the document. The files will be deleted after closing Word. But as typical under Windows, this is done in an insecure way and the data still remains on disk and can be recovered (e.g. by using dedicated recovery software). Using these temporary files, by either intercepting them while the user is working on the document with Word opened or by recovering them from the disk after Word is closed, a parallel depth greater than one is achievable with relative ease. In fact, in the experiments conducted by Filiol [1], up to 4 temporary files were recovered. Any parallel depth above two is more than sufficient for a highly successful cryptanalysis. With an additional semantic validation step even two parallel ciphertexts are usually enough to recover the plaintexts.

Filiol has conducted many experiments for documents in almost all main languages. For example, a set of twenty encrypted 1500-character Word documents, of which the first five had the same password, were used in the detection test, resulting in the output shown in figure 2. The first column

```
z[1-2]  6081 0.658      z[3-15] 4677 0.506      z[6-18]  4611 0.499      z[10-20] 4629 0.501
z[1-3]  6110 0.662      z[3-16] 4604 0.498      z[6-19]  4586 0.496      z[11-12] 4608 0.499
z[1-4]  6141 0.665      z[3-17] 4692 0.508      z[6-20]  4660 0.504      z[11-13] 4670 0.506
z[1-5]  6148 0.666      z[3-18] 4606 0.499      z[7-8]   4647 0.503      z[11-14] 4582 0.496
z[1-6]  4695 0.508      z[3-19] 4605 0.499      z[7-9]   4657 0.504      z[11-15] 4573 0.495
z[1-7]  4636 0.502      z[3-20] 4627 0.501      z[7-10]  4580 0.496      z[11-16] 4582 0.496
z[1-8]  4607 0.499      z[4-5]  6113 0.662      z[7-11]  4594 0.497      z[11-17] 4584 0.496
z[1-9]  4545 0.492      z[4-6]  4634 0.502      z[7-12]  4668 0.505      z[11-18] 4642 0.503
z[1-10] 4638 0.502      z[4-7]  4585 0.496      z[7-13]  4626 0.501      z[11-19] 4591 0.497
z[1-11] 4652 0.504      z[4-8]  4626 0.501      z[7-14]  4550 0.493      z[11-20] 4593 0.497
z[1-12] 4560 0.494      z[4-9]  4622 0.500      z[7-15]  4667 0.505      z[12-13] 4548 0.492
z[1-13] 4682 0.507      z[4-10] 4621 0.500      z[7-16]  4548 0.492      z[12-14] 4592 0.497
z[1-14] 4634 0.502      z[4-11] 4703 0.509      z[7-17]  4642 0.503      z[12-15] 4591 0.497
z[1-15] 4653 0.504      z[4-12] 4627 0.501      z[7-18]  4562 0.494      z[12-16] 4614 0.500
z[1-16] 4578 0.496      z[4-13] 4629 0.501      z[7-19]  4625 0.501      z[12-17] 4574 0.495
z[1-17] 4642 0.503      z[4-14] 4565 0.494      z[7-20]  4629 0.501      z[12-18] 4508 0.488
z[1-18] 4606 0.499      z[4-15] 4664 0.505      z[8-9]   4514 0.489      z[12-19] 4549 0.492
z[1-19] 4601 0.498      z[4-16] 4611 0.499      z[8-10]  4531 0.491      z[12-20] 4619 0.500
z[1-20] 4639 0.502      z[4-17] 4655 0.504      z[8-11]  4617 0.500      z[13-14] 4598 0.498
z[2-3]  6125 0.663      z[4-18] 4537 0.491      z[8-12]  4661 0.505      z[13-15] 4677 0.506
z[2-4]  6126 0.663      z[4-19] 4590 0.497      z[8-13]  4589 0.497      z[13-16] 4646 0.503
z[2-5]  6099 0.660      z[4-20] 4592 0.497      z[8-14]  4709 0.510      z[13-17] 4622 0.500
z[2-6]  4590 0.497      z[5-6]  4625 0.501      z[8-15]  4530 0.490      z[13-18] 4648 0.503
z[2-7]  4633 0.502      z[5-7]  4596 0.498      z[8-16]  4661 0.505      z[13-19] 4655 0.504
z[2-8]  4622 0.500      z[5-8]  4585 0.496      z[8-17]  4703 0.509      z[13-20] 4655 0.504
z[2-9]  4550 0.493      z[5-9]  4521 0.489      z[8-18]  4585 0.496      z[14-15] 4587 0.497
z[2-10] 4651 0.504      z[5-10] 4658 0.504      z[8-19]  4642 0.503      z[14-16] 4562 0.494
z[2-11] 4633 0.502      z[5-11] 4638 0.502      z[8-20]  4694 0.508      z[14-17] 4642 0.503
z[2-12] 4549 0.492      z[5-12] 4540 0.491      z[9-10]  4549 0.492      z[14-18] 4534 0.491
z[2-13] 4643 0.503      z[5-13] 4650 0.503      z[9-11]  4595 0.497      z[14-19] 4651 0.504
z[2-14] 4613 0.499      z[5-14] 4594 0.497      z[9-12]  4593 0.497      z[14-20] 4577 0.495
z[2-15] 4618 0.500      z[5-15] 4633 0.502      z[9-13]  4519 0.489      z[15-16] 4521 0.489
z[2-16] 4651 0.504      z[5-16] 4638 0.502      z[9-14]  4565 0.494      z[15-17] 4613 0.499
z[2-17] 4595 0.497      z[5-17] 4598 0.498      z[9-15]  4660 0.504      z[15-18] 4615 0.500
z[2-18] 4627 0.501      z[5-18] 4500 0.487      z[9-16]  4599 0.498      z[15-19] 4542 0.492
z[2-19] 4708 0.510      z[5-19] 4585 0.496      z[9-17]  4563 0.494      z[15-20] 4728 0.512
z[2-20] 4576 0.495      z[5-20] 4611 0.499      z[9-18]  4627 0.501      z[16-17] 4548 0.492
z[3-4]  6087 0.659      z[6-7]  4649 0.503      z[9-19]  4632 0.501      z[16-18] 4668 0.505
z[3-5]  6150 0.666      z[6-8]  4560 0.494      z[9-20]  4612 0.499      z[16-19] 4645 0.503
z[3-6]  4629 0.501      z[6-9]  4620 0.500      z[10-11] 4594 0.497      z[16-20] 4635 0.502
z[3-7]  4570 0.495      z[6-10] 4571 0.495      z[10-12] 4498 0.487      z[17-18] 4626 0.501
z[3-8]  4583 0.496      z[6-11] 4683 0.507      z[10-13] 4632 0.501      z[17-19] 4579 0.496
z[3-9]  4561 0.494      z[6-12] 4643 0.503      z[10-14] 4604 0.498      z[17-20] 4635 0.502
z[3-10] 4626 0.501      z[6-13] 4661 0.505      z[10-15] 4595 0.497      z[18-19] 4731 0.512
z[3-11] 4644 0.503      z[6-14] 4651 0.504      z[10-16] 4636 0.502      z[18-20] 4663 0.505
z[3-12] 4510 0.488      z[6-15] 4752 0.514      z[10-17] 4596 0.498      z[19-20] 4524 0.490
z[3-13] 4678 0.506      z[6-16] 4573 0.495      z[10-18] 4580 0.496
z[3-14] 4578 0.496      z[6-17] 4547 0.492      z[10-19] 4579 0.496
```

Figure 2: Detection result of Filiol's experiment [1]

denotes which of the documents were compared for parallelism, for example z[1-2] means document number 1 and 2 were compared. The second column lists the number of compared bits and the third column shows the zero bits divided by the total number of bits, as used by the estimator developed in chapter four. The result clearly shows that peaks for files one to five, thus detection of parallel ciphertexts worked very well in this test.

6.5 Testing the Cryptanalysis Algorithm

In further tests to verify the functionality of the cryptanalysis algorithm, multiple texts from different times and backgrounds were considered. These consisted of extracts from Jules Verne novels with a total length of 1200 bytes each, 1500 byte extracts from a speech of the Chief of Staff of the Army held in 2008, containing many technical terms, proper names and diplomatic language and finally extracts from a speech of the president of the French Republic, each 9700 bytes long. In this case the cryptanalysis was done using a multiplicative frequency cumulative function with parameter $a = 0.3$. For further optimization, as discussed in chapter five, overlapping mode, the printable characters only option and a hard decoding were used. The tests were conducted for different depths of parallelism. For a parallelism depth of

- two, about 40% of the plaintexts were recovered.
- three, above 80% of the plaintexts were recovered.
- four and five, above 90% of the plaintexts were recovered.

If soft decoding and semantic analysis were used Filiol [1] would expect a success rate of nearly 100% for the plaintext recovery, when dealing with more than two parallel ciphertexts. A linguist-driven analysis would be necessary to recover all the missing characters in the case of only two parallel ciphertexts.

6.6 The Excel Case

The application of the cryptanalysis technique to Microsoft Excel is more tricky than in the Word case. The detection part of the algorithm is harder because the structure of Excel files is somewhat more complex than the structure of Word documents. For example whenever an Excel document is modified the new content is located at the end of the data, not at the location where the modification occurred. Despite these complications it is, as shown in [1] and [9], still fairly easy to locate the data within the encrypted Excel document. The vulnerability is the same as for Word documents. When encrypting an Excel document the same IV will be used for all modified versions and thus, as long as the users password doesn't change, the key for the RC4 stream cipher will be reused, just like in the Word case. Parallel ciphertexts can then be detected from temporary files or separately saved versions of the same document.

The cryptanalysis part is also somewhat more difficult than in the Word case. Since usually Excel files deal with numerical data, one cannot expect sentences or semantically structured data. Therefore certain optimization features of the cryptanalysis algorithm, such as semantic analysis, might not be possible to the full extend in the Excel case.

doi: 10.2313/NET-2012-04-1_1

This means a very particular corpus, which is specifically constructed to model the context of the Excel spreadsheet, must be used. Usually a higher parallelism depth than in the word case is required. However Excel spreadsheets also offer an advantage over Word documents. Data in Excel files are located between cell separators, which in the binary file are denoted as XX 00 00, where XX is the size of the data inside the next cell [1]. These separators constitute probable plaintext and therefore enable a more efficient recovery process, as described in [3] by the probable word method. According to Filiol, if all these Excel specifics are taken into account, the recovery of Excel plaintexts is as efficient as in the Word case.

6.7 Recap of the Office Attack

Clearly the vulnerability of the Office 2003 encryption lies in the reuse of the IV for revised documents. This leads to key-reuse in these revised documents which form a set of parallel ciphertext files. Through a weakness in the Windows operating system these files can be obtained even after their supposed deletion. This satisfies all the conditions for the application of the cryptanalysis algorithm described in chapter five. The test results show that by means of a language specific frequency analysis of character sequences it is generally possible to recover large parts of the plaintexts. It is possible to further enhance these results, for example by performing a linguist driven analysis subsequently.

The test results show that good results (over 80% recovery) can be expected when at least three parallel documents are obtained. It is also vital to know the language and general context of the texts in order to obtain satisfactory results. Furthermore the attack is only possible if the user does not change their password for every revised version of a document, since IV and password must be reused to produce the same key. It is however very unlikely that a user would change their password after making any modification to the document, meaning the attack would be applicable in most cases.

7. CONCLUSION

Filiol has designed a very interesting and operational technique to detect and break any type of misused or wrongly implemented stream cipher and block cipher in OFB mode. One can imagine many uses for such a technique, that go beyond the simple scenario of an attacker, who wants to detect and break weak ciphertexts. Advanced users might be interested in running such a detection method for parallel ciphertexts against their encrypted documents, in order to make sure that no key reuse has occurred. On the other hand, since this works with any stream cipher, the detection technique can be of interest to companies, who want to employ it as an additional experimental check for the correct implementation of their stream cipher encryption algorithm. Furthermore the key-independent cryptanalysis really shines when one needs to identify and break messages encrypted by an unknown proprietary cipher. Using an USB-key, malware programs or a trojan horse an attacker could possibly detect and gather parallel plaintexts without direct access to the system in question and employ this technique to break them without the need for a time consuming key recovery or any knowledge about the underlying cryptosystem.

Considering the gravity of such an attack, it is surprising that after so many versions of Microsoft Windows and Office, a serious flaw like this still exists in the most widely used office application. However it is worth noting, that Microsoft Office 2007 SP2 and 2010 apparently have experienced a large rework of their security features. These newer versions support any encryption algorithm offered by the Microsoft Cryptographic Application Programming Interface (CryptoAPI), such as AES, DES, ESX, 3DES, and RC2. They also offer a wide selection of cryptographic hashing functions and use much more secure defaults (usually AES with 128-bit key in CBC mode and SHA-1 hashing) than the constant XOR of Office 2003 and previous versions.

Interestingly Microsoft openly acknowledges many of the short comings in previous Office versions in the *Office Document Cryptography Structure Specification* [10]. For example the constant XOR encryption is more fittingly called "XOR Obfuscation". This document also highlights the flaws of the Office RC4 implementation. In fact in this specification it is said that "The Office binary document RC4 CryptoAPI encryption method is not recommended, and ought to only be used when backward compatibility is required" [10]. Not only is the implementation of this encryption method susceptible to the key-independent cryptanalysis described by Filiol but also to several other attacks, as outlined by the specification itself. For example the password may be subject to rapid brute-force attacks, because of the weak key derivation algorithm, using a single hash function instead of an iterated hash, which is recommended by *RFC2898*. Furthermore the key is derived from an input only 40-bits in length, thus the encryption key may be subject to brute force attacks even on current hardware. It is also stated that "some streams might not be encrypted" and "document properties might not be encrypted", which would explain several plain values that were found in [1] and [9] when analyzing the encrypted Office documents. Finally it is also said that "key stream reuse could occur", which is the flaw that allowed for the key-independent cryptanalysis, after detection of the parallel ciphertexts.

Filiol also touches the subject of how such implementation flaws can be used in a trapdoor design. While one flaw alone might not be sufficient for the recovery of plaintexts, it can, when combined with another flaw become a huge security issue and in fact also act as an intended trapdoor. Although the Office case is probably not an intended trapdoor it demonstrates how a combination of flaws can lead to a security problem. 50% of the flaw is at the application level (i.e. incorrect RC4 implementation, reuse of key streams) and the other 50% at the operating system level (i.e. Windows temporary files are insecurely deleted). However it is unlikely that the Office RC4 flaw is an intended trapdoor, since several other serious flaws exist in the Office encryption, which are documented by Microsoft themselves. Still, Filiol's hints on trapdoor design give very interesting insight on how such trapdoors might be build by distributing the security breach over several layers.

Conclusively, it can be said that any product labelled with "X-encryption secure", where X is a well known and approved encryption algorithm or even said to be information theoretically secure (e.g. Vernam system), might in reality

doi: 10.2313/NET-2012-04-1_1

not be secure at all. Implementation flaws or even intended trapdoors, that may consist of flaws distributed across multiple levels and above all misuse of the system by its users may nullify any means of security.

8. REFERENCES

[1] E. Filiol: *How to operationally detect misuse or flawed implementation of weak stream ciphers (and even block ciphers sometimes) and break them - Application to the Office Encryption Cryptanalysis*, Black Hat Europe 2010, April 12-15th

[2] A.J. Menezes, P.C. van Oorschot and S.A. Vanstone: *Handbook of Applied Cryptography*, CRC Press, 1996

[3] C.E. Shannon: *Communication Theory of Secrecy Systems*, Bell System Technical Journal 28, 1949, p. 679-683

[4] C.E. Shannon: *A Mathematical Theory of Communication*, Bell System Technical Journal 27, 1948, p. 379-423 July, p. 623-656 October

[5] A. Papoulis, S.U. Pillai: *Probability, Random Variables, and Stochastic Processes*, 4th Edition, McGraw-Hill Europe, 2002, p. 72-123

[6] E.S. Lee: *Essays about Computer Security*, Centre for Communications Systems Research Cambridge, p. 187

[7] W. Li: *Random Texts Exhibit Zipf's-Law-Like Word Frequency Distribution*, IEEE Transactions on Information Theory, 38(6), 1842-1845, 1992

[8] *Changes in encryption file properties in Office 2003 and Office 2002*, `http://support.microsoft.com/kb/290112`

[9] H. Wu: *The misuse of RC4 in Microsoft Word and Excel*, Preprint IACR, 2005

[10] *Office Document Cryptography Structure Specification*, Microsoft Corporation, 2011

doi: 10.2313/NET-2012-04-1_1

Low Probability, High Stakes: A look at PKI

Alexander Lechner
Betreuer: Dipl.-Inform. Ralph Holz
Seminar Future Internet WS 2011/12
Lehrstuhl Netzarchitekturen und Netzdienste
Fakultät für Informatik, Technische Universität München
Email: lechneal@in.tum.de

ABSTRACT

This paper examines the risk assessment of low probability but extremely high stake events. When evaluating such risks, as for megaprojects or natural disasters, it is most often assumed that the underlying arguments are correct. Especially for events with very low probabilities this naïve approach is not sufficient as the theory and the applied model may be unsound and the calculations may be flawed. New and more complex approaches try to model the uncertainty of unsound arguments in order to provide a more accurate risk estimation of low probability events. One such approach, recently published by Ord *et al.*, is discussed in detail and is further applied to the security of a Public Key Infrastructure (PKI) for which an abstract and generic risk estimation model is developed. The analysis of the model shows that the impact of possible flaws in the argument of a PKI can be vast and must therefore not be neglected. Using several examples of disasters, the underestimation of risks connected to trivial error sources and the necessity of an advanced risk assessment methodology is emphasized.

Keywords

probability theory, risk assessment, security, high stake risks

1. INTRODUCTION

It is often required to assess the risk of highly unlikely events as they may hold extremely high stakes. Examples for such events could be cosmic incidents such as a meteorite colliding with earth, terrorist attacks causing thousands of deaths or success and failure of megaprojects where billions of dollars have been invested.

Based on the low probability of such events it is insufficient to use simple heuristics or the estimated likelihood of the occurrence in order to calculate the risk. It may be more probable that the argument is flawed than that the event actually occurs, which may lead to an underestimation of risk and consequently grave consequences. It is therefore necessary to apply a more sophisticated approach to such risk assessments.

This paper focuses on an advanced risk assessment of low probability high stake events, with an emphasis on the security of PKIs. In section 2 the approach proposed by Ord *et al.* [20] is explained and it is shown how the calculation of the probability can be improved by including the possibility of a flawed argument and by subdividing the argument into the three parts: theory, model and calculation. Further, several considerations for risk assessment are stated and an example of a well-done risk report is given.

Section 3 shows the application of the advanced risk assessment method to the security of Public Key Infrastructures (PKI). A PKI is a well established technique to ensure privacy and identification on the Internet. As such PKIs are widely used by governments, companies and individuals, a failure of such a system bears massive problems [1].

Some examples of well-known disasters in the history of mankind are shown in section 4. The failures are explained and examined with respect to the underlying theory, model and calculation. As many of those failures could have been avoided the necessity of advanced risk assessments becomes obvious.

Finally, section 5 offers a comparison with other approaches and related work, including human risk perception and decision making behaviour.

2. ADVANCED RISK ASSESSMENT

Let us assume the following: A risk report has to be written, assessing the risk of a devastating earthquake taking place in a certain area which has been earthquake-free for centuries. To calculate the probability of an earthquake occurring, a model has to be developed and assessed. Such a model is based on an underlying assumption which may include geodetic and seismologic information as well as geological theories on how earthquakes arise. As earthquakes are extremely rare in the given area, it is highly probable that the result of the model will state a low probability.

Let us use the following notation: "X" the earthquake occurs, "A" the underlying argument is sound and "L" the expected loss in case of the accident. The mathematical formula for calculating the risk is:

$$Risk = P(X) * L \tag{1}$$

A naïve approach would erroneously consider the outcome of the model to be $P(X)$ but in fact the result is $P(X|A)$, as the model can only calculate the probability of X given the underlying argument is valid. By using the naïve approach, the report completely ignores the possibility of the assumption not being sound and may therefore provide a false result. The probability of the argument being flawed may be actually higher than the probability of the earthquake arising.

doi: 10.2313/NET-2012-04-1_2

2.1 Ord et al.'s Approach

The assumption that an argument is entirely correct should never be made, as design and calculation errors are widely spread: section 4 provides several examples of disastrous failures. To address the problem of flawed arguments in risk assessment, Ord *et al.* [20] proposed a more sophisticated approach. Let "X" be the event happening and "A" the assumption being sound, then the probability of X is calculated as follows:

$$P(X) = P(X|A)P(A) + P(X|\neg A)P(\neg A) \tag{2}$$

The naïve approach only considers the first term: $P(X) = P(X|A)$, whereas Ord's approach also considers the case of the argument being unsound. If a naïve risk report states that X is impossible, it only means that $P(X|A) = 0$, yet the error term $P(X|\neg A)P(\neg A)$ may be bigger than zero, leading to a $P(X) > 0$ which means that X is possible after all. It may be impossible to acquire accurate values for $P(\neg A)$ and $P(X|\neg A)$, but coarse estimations can already lead to a change in our risk perception.
Ord's formula can be applied to all risk estimations, but it is of larger importance for low probability events. Having a large $P(X|A)$, the additional error term $P(X|\neg A)P(\neg A)$ is only of little significance in the overall probability estimation. However, for an extremely small $P(X|A)$, the error term may change the resulting $P(X)$ in several orders of magnitude.

A common way of assessing the soundness of an argument is to distinguish between model and parameter uncertainty. Considering the possibility of failures in the background theory and in the calculation, Ord *et al.* [20] proposes a distinction between the argument's theory, model and calculation. Using this distinction, $P(A)$ can be estimated in a more precise way. For calculations their correctness is considered, for theories and models their adequacy. A theory is adequate if it is able to predict the required qualities with a certain precision. For instance Newton's mechanics is an adequate theory for aerodynamic or static problems but is surely inadequate for computations on a quantum mechanical level.
The distinction between theory, model and calculation results in an extended formula. Let "T" be the background theories are adequate, "M" the derived model is adequate and "C" the calculations are correct. As the model is dependent on the theory and the calculations and theories are independent, we get:

$$P(A) = P(T)P(M|T)P(C) \tag{3}$$

The estimation of the individual terms can facilitate the calculation of $P(A)$, but is still of significant difficulty. Below the three parts are separately described and common error sources are depicted.

2.1.1 Theory

The term "theory" used by Ord *et al.* refers to the argument's theoretical background knowledge. This does not only include established and mathematically elaborated principles, but also specific paradigms or best practices. Theories are defined on a high abstraction level, so they are associated with a higher confidence than models.
As many theories are well-established and underlie various models, a flaw in such theories may have an immense impact on the corresponding scientific area and render numerous models and results, at least partially, invalid. Especially in science fields where proofs and evidences are difficult to provide, such as sociology and psychology, or where empirical analysis is unethical, for instance the lethal dose of ionizing radiation, the theories are based on assumptions, extrapolations or even speculations. History shows that many, even well-established theories were flawed. Examples include the phlogiston theory which was used to explain the oxidation processes or Newton's corpuscular theory of light, in which he stated that light consists of small discrete particles. Another example of a flawed theory is the geocentrism: A theory which was assumed to be correct for nearly 2000 years and was the cause for Ptolemy inferring the Epicyclic model and Brahe the Tychonian system.

2.1.2 Model

A model is derived from the background theory and it models the examined phenomena. Various aspects influence the adequacy and the accuracy of such models. As a model is always an abstraction of reality, certain aspects are neglected whereas others are simplified. Too broad or too narrow models may subsequently lead to incorrect predictions. Further problems arise from *parameter uncertainty*, as the model has to be fed with several parameters in order to make a prediction. Those parameters can be imprecise or estimations by themselves. Furthermore, parameters are not always measurable, for instance if human values like trust or sincerity are part of the model. Finally, many phenomena are not well understood and the corresponding models are based on assumptions and approximations. A way to improve risk assessment for such ill-defined phenomena is to combine several models and theories.
The design and development of the model is a highly delicate procedure. Even if the theory and the formulas are correct, an imprecise parameter or an aspect neglected by the model may produce a significant deviation from the correct result. A small model modification, the literal *flap of a butterfly's wings*, can have a large impact on the outcome.

2.1.3 Calculation

The calculations are independent of the argument's theory and model. Still, calculation errors are more common than expected and can lead to the complete failure of the argument. Many different errors are made: accidental errors like forgetting a certain term, collaboration and communication errors when several teams work on the same problem, numerical errors as discretization and floating point errors.
A flawed calculation may have different consequences: There is the possibility that the error has only little effect on the result e.g. a slightly higher inaccuracy. An example of such a case is given in section 4: A bug in a Pentium processor series caused imprecise results for certain input values. Although this may not affect most customers, it may have consequences for scientific simulations and high performance computing. On the other hand it is also possible that a flawed calculation has a large impact on the whole model. For instance, a flaw in the implementation of a cryptographic algorithm may render a whole system insecure. In history, various accidents showed the impact of calculation errors. One prominent example is the Mars Climate Orbiter, which was lost due to a navigation error caused by a trivial miscalculation. The NASA worked with the metric system whereas the navigation software used the imperial system [6].

doi: 10.2313/NET-2012-04-1_2

Despite the fact that calculations are a frequent source of error, it is hard to provide reliable statistics of error rates. Reasons include that calculation errors can be caused by various factors such as hardware failure or human negligence.

2.2 Considerations

Ord *et al.* provide an example of a plausible risk analysis which was performed by Giddings and Mangano in 2008 [8] and uses multiple sub-arguments: will the LHC cause black holes, which subsequently will absorb earth? This is a perfect example of a low probability but extremely high stake risk. To strengthen the argument Giddings and Mangano used 3 different sub-arguments:

- A_1: As a consequence of different physical theories, black hole decay is highly probable.
- A_2: If a non-decaying black hole will be created, it most probably will not be able to discharge itself, which is necessary in order to be harmful.
- A_3: If discharging black holes can be created, the time that is required to consume Earth would still be shorter than Earth's lifespan. This is valid under several assumptions on how black holes interact with matter.

One special problem of this example is that the underlying theories are highly theoretical and difficult to verify. Still, the combined argument consisting of A_1, A_2 and A_3 is significantly stronger than the sub-arguments themselves, as one argument comes into play if the previous one fails. Consequently, the combined error term $P(\neg A_1, \neg A_2, \neg A_3)$ is smaller than the error terms themselves: $P(\neg A_1)$, $P(\neg A_2)$ and $P(\neg A_3)$. Giddings and Mangano state that our current understanding of physics suggests black hole decay A_1 and the inability to discharge themselves A_2 as highly probable. Regarding the uncertainty of the parameters and the assumptions made for the theories, they did not provide a certain probability value for X, instead they concluded that there is no significant risk of black holes.

3. PKI SURVEY

Public Key Infrastructures (PKI) were developed as a consequence of public key cryptography. With the usage of public key cryptography, two parties can confidentially exchange messages and provide authentication by using signatures. For such tasks, both parties have to possess a public encryption key and a private decryption key. Sending a secure message requires the user to encrypt the message with the recipient's public key. The message can then only be decrypted by using the recipient's private key. One of the main issues is to ensure that the other party is the one it claims to be.
A PKI provides a framework for authentication and for the secure exchange of information over an insecure network. Therefore PKI uses certificates issued by Certificate Authorities (CA), which bind public keys to the verified user identities. Such certificates contain at least the identity information, an expiration date and the user's public key. In a hierarchical PKI the certificate of a small CA (e.g. a company CA) can be recursively signed by a larger CA, finally signed by a root CA. So called Registration Authorities (RA) are

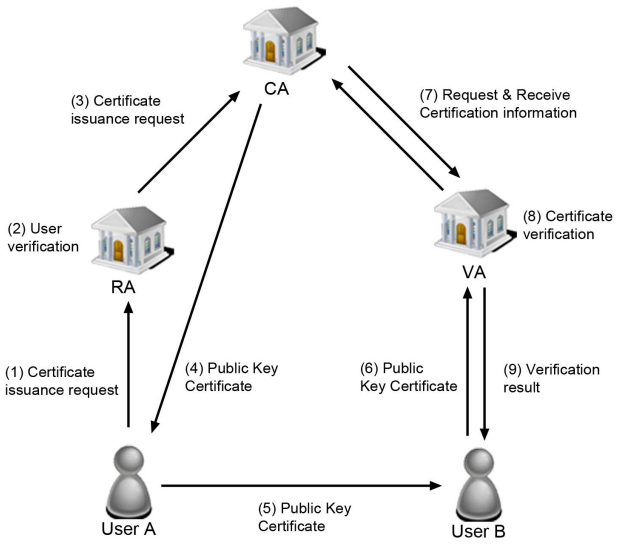


Figure 1: PKI: Certificate issuance and verification.

responsible for the verification of the user identities. The validation of the certificates, namely if a given public key corresponds to the intended user, is performed by Validation Authorities (VA). The issuance and validation procedure can be seen in figure 1. First, user A applies for a new certificate. The RA verifies the user's identity and forwards the request to a CA, which issues the certificate. User A can now proof his identity using the received certificate. Let us now assume that user B wants to check if user A is the person he claims to be. Therefore, user B forwards user A's certificate to a VA, which most often corresponds to the used software e.g. a web browser. The VA uses additional information from the CA, for instance if the certificate is still valid. If the validation process is successful, user B can be sure that user A is the person declared in the certificate. In the following analysis we focus on the X.509 standard, one of the most popular standards used for PKIs. This ITU-T standard specifies the standard formats of certificates, revocation lists and a certification path validation algorithm [4]. There are other approaches which are not covered by this survey. One example is the Web of Trust (WoT) where users express their trust by signing each others certificates and which has the popular implementation PGP.
As a high level of trust is required in many application areas, PKIs must provide an extremely strong security. In the following an advanced probability estimating model for the security of PKI is presented.

3.1 Need for Risk Assessment

Many companies, Web services and individuals rely on PKIs nowadays. Several PKIs exist and application areas include online banking, secured communication and access management. But as PKIs and public key cryptography became widely used, various issues emerged. Such issues range from protocol attacks to the question of confidence in the involved authorities. The end of PKI was often predicted, as it has been criticized by several quarters. However, PKI is not

doi: 10.2313/NET-2012-04-1_2

dead and the criticism helped to evolve the technology [1].
Assuming an ideal world, a highly simplified model and ignoring its fallibility, PKI can be considered highly secure. In fact, in our real world those assumptions do not apply. Perfect cryptographic algorithms and protocols do not exist, governments and politics may influence the trustfulness of CAs and infrastructure components may malfunction.
Ellison and Schneier [7] state 10 major risks of PKIs including "How secure is the verifying computer?", "Is the CA an authority?" and "Is the user part of the security design?". Other issues are mentioned by Adams and Just [1], who describe the evolution and history of PKIs and Boldyreva *et al.*[3], who analyze the security and efficiency of PKIs and provide considerations for building encryption and signature schemes. Besides the numerous cryptographic and computational security issues, the trust in, and the reliability of authorities plays a major role. Authorities do not always deploy PKI correctly. A recent report at the Defcon 18 [21] showed several problems concerning CAs: Certain CAs issued certificates with weak keys, they did not revoke invalid certificates and they signed unqualified names like "localhost" multiple times. Further problems arise from the usage of certificate revocation lists (CRL). If a certificate is no longer valid as it was lost or stolen, the CA has to revoke it. Best practices require to check such CRLs when validating a certificate. However, this practice is often ignored, as the recent DigiNotar hack showed [18]. A poor deployment of PKI results in a higher number of security leaks, facilitating possible attacks by adversaries and substantially undermining the security of the entire infrastructure.
There are obviously many issues concerning PKI but what are the stakes? The stakes highly depend on the usage of PKI. Users may simply use a PKI to communicate with friends, not wanting anyone to be able to eavesdrop. For such a scenario the stakes would be relatively small. More sophisticated usages of a PKI could include classified government information or military communication. The stakes for such scenarios are obviously high. Attackers can use flaws in a PKI for espionage, online fraud or even identity theft. A recent example of a PKI related issue is the Dutch certificate authority DigiNotar, which detected and revoked multiple fraudulent certificates in 2011 [18]. Subsequently the main browsers removed DigiNotar from the list of trusted authorities. As a result the chain of trust for the certificates of the Dutch government's PKI "PKIoverheid", which used DigiNotar certificates for different government operations and agencies, was broken.
As the stakes, depending on the individual usage of PKI, can be extremely high a way of assessing the risk is required. Instead of assuming infallible theories and models and in order to handle the complexity of PKI, the approach introduced by Ord *et al.* [20] was used.

3.2 Probability Model

Let us use the following notation: "A", the argument is sound and "X", a PKI related operation is successfully attacked. To better illustrate the model, we focus on the issuance of a certificate as "X" in this section. The argument is then subdivided in theory, model and calculation.

Theory: The main theory used in a PKI and in cryptographic systems overall is that certain mathematical problems are intractable. Deduced from this theory it is assumed, that the algorithms used for the public key cryptography and protocols in a PKI are secure. However, the security of the public key cryptosystems cannot be guaranteed, as feasible solutions to one of those problems may simply not have been found yet. Furthermore, faster computers and new technologies may influence the infeasibility assumption: Quantum algorithms as for instance Shor's algorithm are able to efficiently solve the discrete logarithm problem.
As the mathematical problems depend on the used public key system, there can be large differences between their complexity and solvability. For instance the discrete logarithm problem on elliptic curves is much more difficult to solve than the factorization of large integers. Some of the most famous existing public key cryptosystems exploit such problems, as for instance RSA, ElGamal and Elliptic curve cryptography (ECC). RSA relies on the infeasibility of factoring large composite integers, ElGamal and ECC rely on the difficulty of computing the discrete logarithm for certain groups.
If the cryptosystem theory is flawed, the impact on $P(X)$ and consequently on PKI will be huge as the PKI can no longer provide the security of its services.

Model: In order to reduce complexity we further subdivided the model into the following parts: Cryptographic Security, Infrastructure Security and Authority Trust.
Cryptographic Security: The cryptographic security refers to the security of the used cryptosystems. As one theory underlies many different models respectively cryptographic algorithms, they differ in many characteristics. Depending on the used parameters, hash functions and (pseudo-) random number generators the security may vary strongly. Attacks exploiting weak conditions or certain circumstances are numerous. For instance plain RSA offers many security leaks which enabled several attacks, such as Coppersmith's attack and chosen-ciphertext attacks [5]. One example of a parameter related issue is the key length of private and public keys. As computing power constantly increases and brute force attacks get feasible, formerly secure keys become insecure. To address this problem, longer keys can be chosen, multiple encryption can be performed (Triple DES) or more secure methods than ECC can be used.
Infrastructure Security: Besides the security of the cryptographic algorithm, the infrastructure bears several additional risks and may render a given PKI insecure. With infrastructure security, the security of all computers, networks and protocols involved in PKI operations as well as their interaction is meant. Especially the interaction between the different components may cause several dangers: as Boldyreva *et al.* [3] state, the mixing of proven cryptographic methods with key registration protocols does not make the system automatically secure. Other security issues concern the involved protocols and servers. Popular attacks, exploiting an insufficient infrastructure security include side channel attacks such as timing and man-in-the-middle attacks.
Authority Trust: One main aspect concerning the security of a PKI is the trust in the involved authorities. Three types of authorities exist: Certificate Authorities (CA) issue digital certificates, Validation Authorities (VA) verify given certificates and Registration Authorities (RA) identify and register certificate holders. The important question is: can an authority be trusted and how can this trust be ensured?

doi: 10.2313/NET-2012-04-1_2

Multiple risks mentioned by Ellison and Schneider [7] address this problem. Further problems arise as a consequence of CAs issuing certificates for other CAs, forming a chain of trust. This procedure requires a global root CA, but in reality there is no such authority. One existing solution is to trust multiple top level CAs, which can be seen as a Bridge CA. This corresponds to the "root store" of web browsers, which consists of a list of trusted CAs. But as there is no hierarchical structure and therefore no root CA the question of trust remains: *Quis custodiet ipsos custodes?* Who can watch the watchmen? Additionally, authorities may be influenced by local laws and governments and be forced to permit them access to their data and to cooperate. Another point is that authorities are no social services but profit oriented companies and therefore interested in a high number of certificates and users. As a result, CAs may not always be an authority on what the certificate contains and the strictness of the user identification and the PKI deployment may be too low [9].

Calculation: The calculation refers to the implementation of every piece of software involved in a certain PKI. This includes the implementation of cryptographic algorithms, protocols and end-user programs such as browsers or mail clients. All of those implementations hold the possibility of errors which can be used to successfully attack a PKI operation. Especially the interaction of different applications, coming from different sources, bears a high risk potential. Due to the mere amount of involved software and the far reaching consequences, it is required to take calculation errors into account: a flaw in the signing implementation may render the whole PKI insecure. Moreover, calculation errors are much more common than one may think. For instance the X.509 standard for PKI had and still has to deal with several implementation issues which can be exploited: Dan Kaminsky demonstrated at 26C3 how to include an unknown attribute into a certificate signing request by using illegal padding or integer overflows [12].

Using the subdivision into theory T, model M and calculation C the probability of the argument A can then be calculated using equation 3. In order to calculate $P(M|T)$, the model has to be adapted to the assessed PKI operation. Depending on the operation, different cryptographic algorithms, protocols and authorities have to be considered.

3.3 Probability Calculation

Having developed a PKI model, the probability of a successful attack $P(X|A)$, given that argument A is sound, can be calculated. But for the calculation of $P(X)$, the possibility of the argument being unsound $\neg A$ has to be taken into account as well. As explained in section 2, the probability of X can then be calculated by using equation: $P(X) = P(X|A)P(A) + P(X|\neg A)P(\neg A)$.
To gain accurate values for $P(A)$ and likewise $P(\neg A)$ is extremely difficult, regarding the argument's complexity and model's the level of detail. Instead, coarse estimations are already sufficient to improve the result significantly, as in the case of PKI there is always a fair chance that the argument is unsound. Possible sources of error in the argument include miscalculated parameters, under-/overestimations of the security and neglected but important factors. As the possibility of a flawed argument is obvious, a reasonable value for $P(X|\neg A)$ is required. This term is even more difficult to calculate, so a rough approximation has to be sufficient [20]. In order to propose a generic approach which is independent of the particular user performing a PKI action, human factors were excluded. If it is desired to calculate the probability of a successful attack depending on the individual, such factors must be taken into account. Examples of human factors influencing the individual security using a PKI are numerous: users may ignore invalid certificates for SSL connection while browsing the Internet, choose easy-to-remember but insecure passwords and, intentionally or unintentionally, not keep their passwords secret. The exploitation of psychology instead of technology can be a much easier strategy for a potential attacker.

3.4 Impact on PKI Security

For PKI and many other cryptographic systems it can be said that the security is only as strong as the weakest link. Despite of having correct cryptographic algorithms, an error in the registration process may still render the whole infrastructure insecure. Already for one single flaw in a PKI there is a relatively high probability that the security of the whole infrastructure can not be longer guaranteed.
In the case of PKI the probability $P(X|\neg A)$ may be much higher than $P(X|A)$, which can lead to far-reaching consequences. Therefore, it is not only recommended but moreover mandatory to consider the fallibility of the argument. As a result, Ord *et al.*'s method provides a much more reliable result compared to the naïve approach, which only considers $P(X|A)$.
In summary it can be said that the security and the risk assessment of a PKI is a complex task. The building of a perfect model can be considered impossible and incorrect models can lead to an extreme underestimation of the risk. As PKI is only a framework, the security should be improved by combining secure and well-proven components. Special attention must be paid to the interoperability of those components, which is a frequent source of error. Although the disproof of the PKI's theory would have the strongest impact on PKI security, a flaw in the model and in the calculations, a poor PKI deployment and the misbehavior of the involved users is by far more probable.

4. FAILURE EXAMPLES

In this section several examples of famous accidents are presented. It can be seen that flaws in design and calculation are wider spread than one may think and the value of the advanced approach by Ord *et al.* becomes clear. The disasters are often accompanied by poor and insufficient risk reports, together with problems in human risk estimation and probability evaluation.

4.1 Therac-25

From 1985 to 1987 the radiation therapy machine *Theriac-25* caused the death of 3 persons and severe injuries of a further 3. Several software errors caused a malfunction of the medical device, giving the patients massive overdoses of radiation. An investigative commission [14] stated, that the prime reason was not a certain software error, but a combination of bad software design and development practices. Those flaws included incorrect synchronization, missing quality reviews and unrealistic risk assessments. The

doi: 10.2313/NET-2012-04-1_2

analysis contained several independent assumptions, such as specific hardware errors with quite low probabilities, but completely ignored software and coding errors as an error source. Even after the first accidents the investigators failed to eliminate the root causes. Instead they continuously fixed individual software flaws which did not solve the main problem.
Splitting this failure up into theory, model and calculation errors it can be said that both model and calculations were flawed. On the calculation side several crucial coding errors were made. Even after the first accidents had happened, the coding errors were not found. This was the result of an overconfidence in software, the lack of independent software code review and no testing of the final software and hardware. On the model side, multiple design errors occurred during the development process. Self-checks, error-detection and error-handling were entirely missing. There was no feature for detecting massive overdoses, the patients reaction was the only indicator on the seriousness of the problem.
Besides the actual errors in design and software, human risk perception and handling was a major issue. Furthermore inadequate software engineering practices were used: software quality assurance practices were violated and a highly complicated design was used.

4.2 Mars Climate Orbiter

The Mars Climate Orbiter (MCO) and the Mars Polar Lander (MPL) were both part of the Mars Surveyor Program established by the NASA in 1994. The aim of the MCO was the study of the Martian climate and atmosphere processes and to act as a communication relay for the MPL. In 1999, after reaching a loose elliptic orbit, the MCO was planned to approach a final 400 km circular orbit. During this orbital insertion maneuver the communication was lost and could not be reestablished. The Mars Polar Lander was destroyed several months later in the course of the landing procedure. The disaster caused a loss of $327.6 million in total [11].
The Phase 1 report released by the Mars Climate Orbiter Mishap Investigation Board [22] concluded that the primary cause of the failure was human error. The navigation software of the MCO, developed by Lockheed Martin, used imperial units (pound-seconds) for the calculation of the momentum whereas the NASA used metric units (newton-seconds). Consequently, the effect on the spacecraft trajectory was underestimated by a factor of 4.45.
The error causing the destruction of the MPL could not be determined. The most likely causes include incorrect sensor feedback and a failure of the heat shield.
The destruction of the MCO can be seen as both a calculation and a model error. The main error was the usage of imperial instead of metric units in a software file. Still, Euler *et al.*[6] state that the error was caused by mistakes in judgment and poor application of standard practices. As an example he names the "Faster, Better, Cheaper" philosophy of the NASA at that time, when all non-value adding activities were eliminated and development steps were reduced.

4.3 Pentium FDIV Bug

The Pentium FDIV bug was a bug in the Intel P5 Pentium floating point unit which led to incorrect results of floating point divisions with certain input values. The error occurred extremely rarely and remained therefore undetected for 1.5 years. At that time, approximately 3-5 million copies of such flawed Pentium processors were in use. Later it came to light that Intel had already been aware of the flaw for several months. In the end, Intel was forced by public pressure to offer a replacement of all affected Pentium processors, resulting in an additional financial burden of $475 million [19].
The flaw was detected by Thomas Nicely [19] in 1994 and is a perfect example of a calculation error, namely a clerical mistake: 5 entries in the look-up tables used for the division algorithm contained corrupt values: a "0" instead of a "2". As a result the error only occurred when the corresponding table entries were used for the calculation. Intel states in its white paper that the worst possible inaccuracy occurs it the 4th significant decimal digit and the probability of a flawed divide instruction with two random input values is 1 in 9 billion [10]. One example of a flawed calculation is given by Nicely [19]:

$$\frac{4195835.0}{3145727.0} = 1.3338204491362410025 \text{ (Correct value)} \quad (4)$$

$$\frac{4195835.0}{3145727.0} = 1.3337390689020375894 \text{ (Flawed Pentium)} \quad (5)$$

Although the error occurs extremely rarely and has nearly no impact on an ordinary end-user, high-level applications which require a high precision may be severely influenced. Encountering the flaw in a scientific, financial or military application, where high precision is mandatory, can entail massive consequences. The stakes of a miscalculation in a financial simulation or a military device can be imagined by anyone.

5. RELATED WORK

In this section we present some of the related research literature which addresses the security of PKIs, risk assessment of low probability high stake events and human risk perception.
One model of assessing the trust in a PKI was developed by Marchesini and Smith [16]. Instead of assuming an ideal world, they adapted the model to the real, imperfect world. To do so, they had to handle many real-world PKI concepts, such as authorization, trust management and certificate revocation. By applying their calculus to several PKI systems they showed its ability to reason about real-world PKIs.
A document issued by the Department of Finance and Deregulation of the Australian Government provides a valuable overview of a PKI risk assessment methodology [2]. They arrange possible threats in 7 different categories, not only referring to infrastructure failures but also to the individual usage of certificates and to social engineering. They further provide a template for risk assessment: Each encountered risk is assigned a likelihood indicator, from "rare" to "almost certain" and a consequence indicator, ranging from "insignificant" to "catastrophic". By combining the likelihood with the impact, a risk analysis matrix is created, showing the significance of the risk. This is a highly simplified and subjective way of performing a risk assessment, but nonetheless valuable as many security leaks are considered and subsequently managed.
As risk analysis is vitally important for companies and scientific projects, it is crucial for risk assessments to consider the psychology of high stakes decision making and human risk perception. Individuals are influenced in many different

doi: 10.2313/NET-2012-04-1_2

ways by social and private factors. This is especially the case for low probability events where the necessary experience is missing. Kunreuther *et al.*[13] discusses low probability and high stake decision making. He states several problems in human risk assessment, like people who insufficiently use or even ignore available information. Further, the likelihood of an event happening is considered so low, that the event is neglected although the stakes are high. Another problem of human risk perception is the focus on short time horizons. Instead of taking the long term consequences into consideration, people tend to focus on a short subsequent time frame. Another important point is the strong influence of feelings and emotions. Having a certain personal experience or a certain aversion can severely change one's own risk estimation. As people miss the experience with low probability decisions, they also lack experience in handling them. As a consequence they may be strongly influenced by social norms or simply decide to take no decision at all. Finally Kunreuther makes several proposals on how to improve human risk perception. He gives two considerations: First, the usage and understanding of prescriptive heuristics has to be thought to humans so that extremely low probabilities can be better interpreted. Secondly, financial incentives should be developed in order to attract companies and governments to consider long term impacts of their actions.

A popular example of low probability, high consequence events are aviation accidents. Although they occur very infrequently, extensive effort is dedicated to reduce and eliminate the probability. In order to assess the risk of such accidents, Luxhoj and Coit [15] presented an Aviation System Risk Model (ASRM). Their model was designed to cover multiple causalities of known and unknown risk factors, as they are often neglected by similar models.

Finally, the book by Morgan and Henrion [17] gives a detailed overview on uncertainty analysis and presents several approaches on how to model and assess probability events. Apart from the philosophical background they also cover mathematical models of uncertainty calculation and propagation and provide information on human judgement about and with uncertainty.

6. CONCLUSION

As the number of megaprojects and the synergy between different areas increases, viable risk assessment is required. Therefore it is mandatory to cover the chance of an unsound assumption by the risk analysis. The argument used for the assessment can be further strengthened by using several independent arguments based on different models and theories.

The methodology, proposed by Ord *et al.*, was applied to PKI in order to provide an overview of its security and to depict possible error sources. It was shown that the advanced risk analysis considerably differs from the naïve approach. Although it is most difficult to gather reliable values for the different probabilities, the consideration of flawed arguments improves the reliability of the resulting probability evaluation significantly.

Further, the analysis of the PKI security showed the value of Ord's *et al.* approach when dealing with low probability, high stake events. The possibility of a flawed argument must not be neglected by PKI risk assessments, as the security primarily relies on the correctness of the theory, the model and the calculations. PKIs are used for highly classified information and are involved in various application areas. The failure of such a PKI may have disastrous impacts. As many critical issues and possible error sources were shown, further research in this area is required to help the existing frameworks and implementations to evolve.

7. REFERENCES

[1] C. Adams and M. Just. PKI: Ten years later. In *3rd Annual PKI R&D Workshop*, pages 69–84, 2004.

[2] Australian Government: Department of Finance and Deregulation. Gatekeeper PKI Framework, Threat and Risk Assessment Template. http://www.finance.gov.au/e-government/security-and-authentication/gatekeeper/index.html, 2010. [Online; accessed 29-August-2011].

[3] R. Boldyreva, M. Fischlin, A. Palacio, and B. Warinschi. A closer look at PKI: Security and efficiency. In *Proc. 10th international conference on Practice and theory in public-key cryptography*, pages 458–475, 2007.

[4] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk. RFC 5280 - Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. Technical report, May 2008.

[5] D. Coppersmith. Small Solutions to Polynomial Equations, and Low Exponent RSA Vulnerabilities. *Journal of Cryptology*, 10:233–260, 1997.

[6] S. D. J. Edward A. Euler and H. H. Curtis. The Failures of the Mars Climate Orbiter and Mars Polar Lander: A Perspective from the People Involved. In *Proc. of Guidance and Control 2001, American Astronautical Society (AAS)*, pages 01–074, 2001.

[7] C. Ellison and B. Schneier. Ten Risks of PKI : What You're not Being Told about Public Key Infrastructure. *Computer Security Journal*, XVI(1):1–8, 2000.

[8] S. B. Giddings and M. L. Mangano. Astrophysical implications of hypothetical stable TeV-scale black holes. *Physical Review D*, 78(3), Aug. 2008.

[9] R. Holz, L. Braun, N. Kammenhuber, and G. Carle. The SSL landscape - a thorough analysis of the X.509 PKI using active and passive measurements. In *Proc. 11th Annual Internet Measurement Conference (IMC '11)*, 2011.

[10] Intel Corporation. Statistical analysis of floating point flaw. http://www.intel.com/support/processors/pentium/fdiv/wp. [Online; accessed 29-August-2011].

[11] D. Isbell. 1998 Mars Missions. ftp://ftp.hq.nasa.gov/pub/pao/presskit/1998/mars98launch.pdf, 1998. [Online; accessed 29-August-2011].

[12] D. Kaminsky. Black Ops of PKI, Talk at 26th Chaos Communication Congress. http://events.ccc.de/congress/2009/Fahrplan/events/3658.en.html, 2009. [Online; accessed 29-August-2011].

[13] H. Kunreuther, R. Meyer, R. Zeckhauser, P. Slovic, B. Schwartz, C. Schade, M. F. Luce, S. Lippman, D. Krantz, B. Kahn, and R. Hogarth. High Stakes Decision Making: Normative, Descriptive and

doi: 10.2313/NET-2012-04-1_2

Prescriptive Considerations. *Marketing Letters*, 13:259–268, 2002.

[14] N. G. Leveson. An Investigation of the Therac-25 Accidents. *IEEE Computer*, 26(7):18–41, 1993.

[15] J. T. Luxhoj and D. W. Coit. Modeling low probability/high consequence events: an aviation safety risk model. In *Proc. of the annual Reliability and Maintainability Symposium (RAMS)*, pages 215–221. IEEE Computer Society, 2006.

[16] J. Marchesini and S. W. Smith. Modeling Public Key Infrastructure in the Real World. In *2nd European PKI Workshop: Research and Applications*, pages 1–17, 2005.

[17] M. Morgan and M. Henrion. *Uncertainty: A Guide to Dealing with Uncertainty in Quantitative Risk and Policy Analysis*. Cambridge University Press, 1992.

[18] Mozilla Security Blog. DigiNotar removal follow up. http://blog.mozilla.com/security/2011/09/02/diginotar-removal-follow-up, 2011. [Online; accessed 25-October-2011].

[19] T. R. Nicely. Pentium FDIV flaw. http://www.trnicely.net/pentbug/pentbug.html, 2011. [Online; accessed 29-August-2011].

[20] T. Ord, R. Hillerbrand, and A. Sandberg. Probing the improbable: methodological challenges for risks with low probabilities and high stakes. *Journal of Risk Research*, 13(2):191–205, March 2010.

[21] J. B. Peter Eckersley. Is the SSLiverse a safe place? Talk at 27C3. http://www.eff.org/files/ccc2010.pdf, 2010. [Online; accessed 29-August-2011].

[22] A. Stephenson. Mars Climate Orbiter Mishap Investigation Board Phase 1 Report. *NASA*, 10, November 1999.

doi: 10.2313/NET-2012-04-1_2

High Performance Client Server Infrastructure

Tobias Höfler
Betreuer: Philipp Fehre
Seminar Future Internet WS2011
Lehrstuhl Netzarchitekturen und Netzdienste
Fakultät für Informatik, Technische Universität München
Email: hoeflert@in.tum.de

KURZFASSUNG

Die Client-Server Architektur ist trotz ihres Alters noch immer aktuell. Seit den Anfängen des Internets und insbesondere heute steigt die Anzahl an Clients immens schnell, was hochperformante und skalierbare Architekturen notwendig macht. Mögliche Ansätze, Client-Server Architekturen effizient zu realisieren werden in diesem Paper, mit den entsprechenden Grundlagen, behandelt.

Schlüsselworte

Sockets, I/O Modelle, Event, Node, iterativer Server, paralleler Server

1. EINLEITUNG

Das Client-Server Modell ist das vorherrschende Modell in verteilten Anwendungen. Hierbei bietet der Server gewisse Dienste und Daten an, während Clients diese in Anspruch nehmen. Somit besteht zwischen diesen beiden eine Produzenten-Konsumenten-Beziehung. Die immens wachsende Zahl an Teilnehmern in verteilten Anwendungen, nicht nur durch Computer sondern auch durch mobile Endgeräte, lässt die Zahl an Clients kontinuierlich wachsen, was eine effiziente Implementierung der Server notwendig macht. Dieses Thema wird in der vorliegenden Arbeit behandelt. Schon im Jahr 2003 beschäftige sich Dan Kegel mit diesem Problem ([6]), wobei dieses auch acht Jahre später nicht an Bedeutung verloren hat.
Kapitel 2 bildet die notwendigen Grundkenntnisse für die weiteren Kapitel. Kapitel 3 diskutiert danach mögliche Entwurfskonzepte für Client-Server-Architekturen mit deren Vor- und Nachteilen. In Kapitel 4 wird eine Einführung in die ereignisorientierte Programmierung gegeben inklusive der Vorstellung eines aktuellen Vertreters der dieses Paradigma verfolgt. Im letzten Kapitel 5 werden mit Beispielimplementierungen Geschwindigkeitstests durchgeführt und ausgewertet.

2. SOCKET PROGRAMMIERUNG

Zur Netzwerkprogrammierung werden typischerweise Sockets verwendet. Ursprünglich, für die erste BSD-Implementierung entwickelt, stellten diese die Grundlage für Implementierungen anderer Betriebssysteme dar. Sockets ermöglichen es Anwendungen über Transportschichtprotokolle wie TCP oder UDP miteinander über ein Netzwerk zu kommunizieren. Der Zugriff auf die Sockets selbst wird über Funktionen einer Programmbibliothek realisiert. Sie implementiert Funktionen wie *socket()* um einen Socket zu öffnen, *bind()* um den Socket an eine Adresse und einen Port zu binden, *listen()* um auf dem Rechner auf ankommende Verbindungen zu lauschen, *connect()* um eine Verbindung zu einem anderen Socket aufzubauen, *write()* um auf den Socket zu schreiben und um somit Daten an den Kommunikationspartner zu senden, *read()* um Daten vom Socket zu lesen und *close()* um eine Verbindung abzubauen. Der Socket selbst wird hierbei vom Betriebssystem bereitgestellt. Über diese Funktionen hinaus, existieren noch einige andere Funktionen die die Möglichkeiten mit Sockets zu interagieren erweitern. Auch werden verschiedene Sockettypen unterschieden. Der prinzipielle Ablauf einer Socket-Kommunikation unterscheidet sich, abhängig von der jeweiligen Rolle des Prozesses, das heißt ob dieser die Client- oder Serverseite realisiert. Eine Kommunikation mittels TCP bedeutet zum Beispiel für den Server, den Aufruf der Funktionen socket(), bind() und listen() um einen Socket zu kreieren, diesen an eine Adresse und einen Port zu binden und um auf eingehende Verbindungen zu warten. Die Clientseite ruft lediglich ein socket() und ein connect() auf um einen eigenen Socket zu erzeugen und eine Verbindung zum Server herzustellen. Nach einem serverseitigen accept() kann die eigentliche Kommunikation mittels read() und write() erfolgen. Abschließend rufen beide Prozesse die close() Funktion auf, um die jeweiligen Sockets zu schließen[4].
Die bisher eingeführten Funktionen und der schematische Kommunikationsablauf bilden die Basis für die folgenden Kapitel.

2.1 I/O Modelle

Der Zugriff auf Sockets aus der Anwendung heraus erfolgt stets mit Hilfe von Deskriptoren. Diese identifizieren den vom Betriebssystem erstellen Socket eindeutig und bieten somit die Kommunikationsschnittstelle zwischen Anwendung und Socket. Beim Zugriff auf diese, kann ein Prozess blockieren. Im Hinblick auf diese Blockierung werden im Folgenden fünf unterschiedliche Methoden für den Zugriff auf Sockets vorgestellt.

2.1.1 Blockierende I/O

Standardmäßig blockieren Sockets. Man spricht hierbei vom *blockierenden I/O* Modell. Dies bedeutet, dass ein Prozess der beispielsweise die Funktion *read()* aufruft, um Daten von einem Socket zu lesen blockiert, das heißt mit seiner Ausführung stoppt und so lange wartet, bis der Empfang der Daten abgeschlossen ist. In der Zwischenzeit kann der Prozess keinerlei weitere Aktionen durchführen.

doi: 10.2313/NET-2012-04-1_3

2.1.2 Nichtblockierende I/O

Dem Gegenüber steht die *nichtblockierende I/O*. Hierbei muss ein Socket explizit als „nicht blockierend“ gesetzt werden. Ruft ein Prozess nun die *read()*-Funktion in diesem Modell auf, wird er nicht blockiert, sondern bekommt vom Betriebssystem-Kernel eine Fehlermeldung zurück. Diese Fehlermeldung impliziert, dass die angeforderten Daten noch nicht vollständig übertragen wurden, somit dem Prozess noch nicht zur Verfügung stehen. Der Prozess wiederholt nun die Anfrage mithilfe von *read()* so lange, bis keine Fehlermeldung zurückgegeben wird, sondern ein „OK“. Damit sind die Daten übertragen und für den Prozess nutzbar. Diese wiederholte Anfragen wird auch *Polling* genannt. Einer der größten Nachteile ist hierbei der (unnötige) Verbrauch von CPU Zeit, also Rechenleistung.

2.1.3 I/O Multiplexing

Das dritte Modell ist das *I/O Multiplexing*. Dieses bezieht sich auf die *poll*, *select* und *epoll* Systembefehle welche grundsätzliche die gleiche Funktionalität liefern. Diese Funktionen blockieren genauso wie es bei dem *blockierenden I/O* Modell der Fall war. Im Unterschied zu den dort vorgestellten Mechanismen, werden den oben genannten Funkionen allerdings nicht nur ein Socket-Deskriptor, sondern gleich mehrere übergeben. Der jeweilige Befehl blockiert dann solange bis einer der registrierten Sockets bereit zum lesen oder schreiben ist. Das I/O Multiplexing bietet somit keine Vorteile, wenn der Prozess nur einen Socket- bzw. I/O-Deskriptor, sondern nur wenn er mehrere gleichzeitig verwaltet, ohne dass einer den kompletten Ablauf blockiert.
poll und *select* wurden etwa zur gleichen Zeit unabhängig voneinander entwickelt. *select* in BSD und *poll* in System V. *epoll* wurde später im Linux Kernel implementiert und ist vor Allem auf Skalierbarkeit optimiert[2].

2.1.4 Signalgesteuerte I/O

Das *signalgesteuerte I/O Modell* bietet eine weitere nichtblockierende Möglichkeit für den Zugriff auf Deskriptoren. Grundsätzlich verursachen Signale eine Prozessunterbrechung. Hat dieser Prozess eine Routine für das entsprechende Signal registriert, wird diese ausgeführt und der Prozess fortgeführt. Falls dies nicht der Fall ist, führt der Betriebssystemkernel die Standardroutine des jeweiligen Signals aus[9]. Im Kontext der Socket-Programmierung kann das *SIGIO*-Signal verwendet werden. Der Betriebssystem-Kernel informiert den Prozess über dieses Signal, sobald der Socket lese- bzw. schreibbereit ist. Hierzu registriert der Prozess eine Routine, die das *SIGIO*-Signal abfängt und die Daten vom Socket liest oder diesen beschreibt. Dies bedeutet, dass keine Blockierung stattfindet, solange der Socket nicht zum lesen oder schreiben bereit ist.

2.1.5 Asynchrone I/O

Bisher wurde das Lesen eines Sockets als einfache Operation angesehen. Tatsächlich geschieht jedoch noch ein Schritt zwischen dem Warten auf das vollständige Ankommen der Daten und deren Verfügbarkeit für den Prozess. Nachdem die Daten angekommen sind, müssen diese zuerst vom Kernel in den Userspace kopiert werden, damit der Prozess darauf zugreifen kann. Die bisherigen Methoden haben stets den Prozess informiert, sobald die Daten angekommen und im Kernelspace verfügbar sind, das Kopieren der Daten in den Userspace musste jedoch der Prozess selbst durchführen, was wiederum eine blockierende Aktion darstellt. Das *asynchrone I/O Modell* optimiert diesen Sachverhalt, indem der Prozess erst informiert wird, sobald die Daten vollständig angekommen und vom Kernel- in den Userspace kopiert sind. Somit ist dies ein Modell, was die komplette Interaktion mit Sockets nicht blockierend realisiert. Leider wird dieses Modell lediglich von einigen POSIX.1 Systemen unterstützt[7].

2.2 Zusammenfassung & Bewertung

Die hier vorgestellten Möglichkeiten mit Sockets zu interagieren bilden die Grundlage einer jeden Netzwerkprogrammierung. Falls diese Aktionen selbst nicht durch die verwendete Programmiersprache gekapselt werden, können hierbei durch geschickte Wahl des Modells Performancesteigerungen erzielt werden. Grundsätzlich werden blockierende und nichtblokierende I/O Modelle unterschieden, wobei der Prozess beim erstgenannten vollständig anhält um auf Daten zu warten. Für effiziente Implementierungen mit mehreren Clients ist grundsätzlich ein nichtblockierender Ansatz sinnvoll. Weitere Möglichkeiten der Interaktion mit Sockets bietet das I/O Multiplexing und die signalgesteuerte I/O. Bei beiden Modellen wird der Prozess informiert, sobald der Socket-Deskriptor bereit ist. Das signalgesteuerte Modell enthält leider einige Hürden für effiziente Implementierungen, da das *SIGIO-Signal* bei einigen Sockets nicht nur gesendet wird, sobald die Daten vollständig angekommen sind, sondern auch wenn diverse andere Kommunikationsereignisse auftreten. Das asynchrone I/O Modell steigert die Performanz noch um ein Weiteres, indem hierbei der aufrufende Prozess erst informiert wird, sobald die Daten vollständig für ihn verfügbar sind.

3. CLIENT-SERVER DESIGN

Wie in Kapitel 1 erwähnt, bedient ein Server typischerweise mehrere Clients. Dabei lauscht er immer auf eingehende Verbindungen (vgl. 2), bearbeitet diese und schickt dem entsprechenden Client eine Antwort. Hierbei hat die Art der Bearbeitung einer Anfrage enorme Auswirkungen auf die Effizienz des Server. Mögliche Arten werden im Folgenden beschrieben.

Die einfachste Art einen Serverprozess zu implementieren ist der sequentielle Fall. Hierbei bearbeitet der Server die Anfrage eines Clients, schickt die Antwort an diesen zurück und lauscht anschließend wieder auf neue Verbindungen. Diese Art von Server wird auch *iterativer Server* genannt wobei die ankommenden Anfragen nacheinander beantwortet werden.
Dem gegenüber steht der *parallele Server*. Er kreiert für jede einzelne Client-Anfrage mittels *fork()* einen neuen Kind-Prozess der die Anfrage dann bearbeitet und die Antwort an den Client schickt. Der Hauptprozess kann dabei sofort nach dem Erzeugen des neuen Prozesses neue Anfragen annehmen. Die Anzahl an Kind-Prozessen ist dabei ausschließlich durch das Betriebssystem begrenzt bzw. durch den verfügbaren Speicher.
Abbildung 1 zeigt beide Servertypen im Pseudocode. Ganz links ist der iterative Server, welcher eine Clientanfrage erhält, diese bearbeitet und die entsprechende Antwort zurück sendet. In der Mitte ist die Implementierung des parallelen Servers skizziert. Er erhält eine Anfrage des Clients, erzeugt

doi: 10.2313/NET-2012-04-1_3

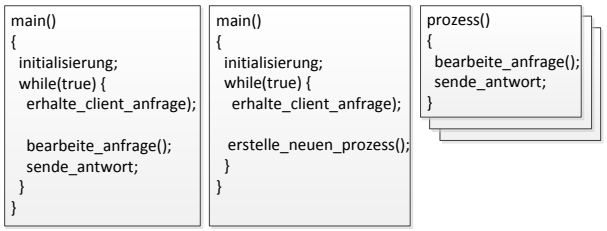


Abbildung 1: Iterativer vs. Paralleler Server (vgl. [1], S. 38)

einen neuen Prozess und kann sofort wieder neue Clientanfragen annehmen. Der erzeugte Prozess, rechts in der Abbildung, bearbeitet die Anfrage und sendet die Antwort an den Client.
Grundsätzlich ist bei mehreren Clients die Antwortzeit paralleler Server besser als die bei iterativen. Hinzu kommt hierbei allerdings die Zeit, die für das *fork()*, also die Erzeugung des neuen Kind-Prozesses entsteht bzw. allgemein die für die Prozesssteuerung notwendig ist. Diese entfällt natürlich beim *iterativen Server*[7].
Um einen *parallelen Server* effizient zu implementieren, muss die zusätzliche benötigte Zeit gegenüber dem *iterativen Server* möglichst klein sein. Aus diesem Grund werden häufig Threads anstatt Prozesse verwendet. Diese Thematik wird im Folgenden behandelt.

3.1 Prozesse und Threads

„Ein Prozess ist ein Programm in Ausführung“. Er ist Grundbaustein der Parallelität auf Rechnern. Hierzu wird ein Zeitmultiplexing der CPU durchgeführt, die, kontrolliert vom Betriebssystem, verschiedenen Prozessen Rechenzeit zuteilt. Um einen Prozess zu verwalten, nutzt das Betriebssystem den *Process Control Block (PCB)*, der alle notwendigen Informationen über den Prozess enthält. Hierzu zählen beispielsweise eine eindeutige Kennung, Prozesszustand, Prozesskontext und einige mehr. Wechselt das Betriebsystem nun den Prozess, dem aktuell die komplette Rechenleistung zugeteilt ist, müssen einige Schritte durchgeführt werden. Zuerst muss der Kontext des aktuell laufenden Prozesses gespeichert werden, das bedeutet dessen Informationen in einen PCB geschrieben werden. Dann wird anhand gewisser Algorithmen ein neuer Prozess ausgewählt (Scheduling). Der letzte Schritt besteht darin, die Informationen des PCBs des neuen Prozesses in die CPU zu kopieren[4]. Diese Prozessverwaltung kostet Ressourcen und Zeit, weshalb häufig Threads anstatt Prozesse für parallele Server verwendet werden. Ein Thread ist eine Aktivität innerhalb eines Prozesses. Ein Prozess beinhaltet immer mindestens einen Thread, wobei mehrere Threads in einem Prozess laufen können. Diese teilen sich eine gemeinsame Prozessumgebung, liegen also im selben Adressbereich und geöffnete Datei- oder Socket-Deskriptoren, Signale und einiges mehr. Sie besitzen jedoch einen eigenen Programmzähler, eigene Register und einen eigenen Stack. Da diese Informationen jedoch weitaus geringer sind, als die eines Prozesses, ist die Erzeugung eines Therads zehn bis hundert mal schneller als die eines Prozesses und die CPU kann deutlich schneller zwischen Threads umschalten, als zwischen Prozessen[12]. Deshalb werden Threads auch als leichtgewichtige Prozesse bezeichnet.
Zur Realisierung von Threads gibt es mehrere Möglichkeiten: Sie können auf Benutzer- oder Betriebssystem- bzw. Kernelebene oder zwischen diesen beiden Ebenen laufen. Bei der erstgenannten Methode übernimmt eine Bibliothek die Verwaltung der Threads. Somit ist das Betriebssystem außen vor und erkennt die Existenz mehrerer Threads nicht. Diese Variante ist einfach zu implementieren und besitzt einen Geschwindigkeitsvorteil gegenüber der zweiten Methode, da nie zwischen Benutzer- und Systemmodus umgeschaltet werden muss. Des weiteren hat eine große Anzahl an Threads kaum belastende Auswirkungen auf das Betriebssystem da dieses prinzipiell nichts von den Threads weiß, folglich auch keine Verwaltung übernehmen muss. Diese Methode besitzt allerdings auch Nachteile. So ist das Scheduling, also die Verteilung der CPU-Zeit auf die Threads nicht fair. Das Betriebsystem führt das Scheduling hierbei auf Prozessebene durch. Besitzt ein Prozess nun beispielsweise deutlich mehr Threads als ein anderer, bekommen trotzdem beide die gleiche Rechenzeit zugeteilt. Auch eine Priorisierung der Threads ist nicht möglich. Vorteile eines Mehrprozessorsystems können nicht ausgenutzt werden, da die Thread-Bibliothek diese nicht zur Kenntnis nehmen kann. Dies hat zur Folge, dass Threads in einer solchen Umgebung niemals echt parallel ablaufen können, sondern lediglich die Prozesse, in denen sie laufen.
Die zweite Variante ist die Nutzung von Threads auf Betriebssystemebene. Hierbei werden diese vom Betriebssystem selbst verwaltet, folglich fallen die eben genannten Nachteile der ersten Methode weg, was bedeutet, dass nun ein faires Scheduling stattfindet und auch eine Priorisierung vorgenommen werden kann. In einem Mehrprozessorsystem oder einer CPU die hardwareseitig Multithreading unterstützt kann nun auch eine echt paralleler Ablauf von Threads stattfinden.
Diesen Vorteilen stehen die Nachteile gegenüber, dass zwischen System- und Benutzermodus gewechselt werden muss, wenn ein neuer Thread angelegt wird. Weiterhin können sehr viele Threads das Betriebssystem belasten.
Die dritte Möglichkeit für Threads ist ein hybrider Ansatz, welcher im Solaris Betriebssystem implementiert ist. Hierbei werden Benutzer-Threads, Kernel-Therads zugeteilt, womit dieser Ansatz die Vorteile beider erstgenannten Methoden besitzt[1].

3.2 Threads - Vor- und Nachteile

In Kapitel 3.1 wurden hauptsächlich Vorteile von Threads gegenüber Prozessen dargestellt. Wie Kapitel 5 und auch [7] anhand verschiedenster Messungen belegt, sind Threadimplementierungen schneller als Prozessimplementierungen. Neben den in Kapitel 3.1 genannten Nachteilen existieren jedoch noch einige mehr.
Wie bereits erwähnt, laufen Threads innerhalb eines Prozesses im gleichen Adressraum. Wenn mehrere Threads parallel laufen, was bei einer Implementierung eines parallelen Servers der Fall ist, kann der Programmierer nicht vorhersagen, welcher Thread gerade läuft. Beim Zugriff auf gemeinsame Ressourcen können somit unvorhersehbare Ereignisse eintreten, weshalb Threads synchronisiert werden müssen. Diese Aufgabe kann unter Umständen schwierig werden und erfordert eine sorgfältige Programmentwicklung[9]. Die Parallelität des Servers macht das debuggen, also die Lokalisierung von Fehlhandlungen, auch schon bei einfachen Aktionen komplex. Auch das Testen der Implementierung ist sehr komplex, da eintretende Aktionen oder Fehler vom

doi: 10.2313/NET-2012-04-1_3

Scheduling abhängen, somit auch schwierig reproduzierbar sind. Durch die Nutzung von Threads auf Benutzerebene, ist die Portierung der Applikation auf andere Systeme sehr umständlich, da die jeweiligen Bibliotheken Betriebssystem-spezifisch sind.
Grundsätzlich macht die Verwendung von Threads die Programmentwicklung sehr komplex und ist häufig Quelle von Fehlern.[8].

3.3 Preforking und Prethreading

Ein Ansatz, zur weiteren Optimierung der Serverarchitektur ist das Preforking bzw. Prethreading. Hierbei wird, wie zu Beginn dieses Kapitels, ein Child Prozess bzw. ein Thread pro Client erzeugt. Hierbei gingen die Kosten dieser Erzeugung zu Lasten der Antwortzeit, da die Erzeugung erst mit einer Client-Anfrage durchgeführt wird. Diese Zeit, kann jedoch auch auf den Serverstart verlagert werden. Hierzu wird eine gewisse Anzahl an Prozessen bzw. Threads beim Start des Serverprozesses erzeugt. Diese können dann sofort neue Client Anfragen bearbeiten, ohne weitere Kosten zu verursachen. Im Folgenden wird lediglich auf die Implementierung mittels Threads eingegangen, da diese in der Regel schnellere Serverimplementierungen ermöglichen. Abbildung 2 zeigt ein solches Prethreading Szenario. Hierbei wurden n

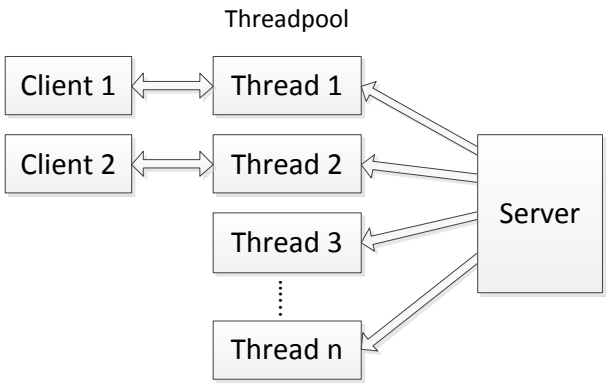


Abbildung 2: Prethreading (vgl. [7], S. 728)

Threads erzeugt wobei aktuell zwei Clients verbunden sind. Im Threadpool sind noch n-2 weitere Threads vorhanden, somit können problemlos n-2 weitere Clients bedient werden. Bei mehr als n Clients muss eine effiziente Server Implementierung weite Threads erzeugen bzw. die Notwendigkeit der neuen Erzeugung vorhersehen und schon durchführen, bevor alle Threads im Pool durch Clients belegt sind[7].

3.4 Zusammenfassung

Grundsätzlich werden iterative und parallele Serverimplementierungen unterschieden. Während der iterative Server Anfragen sequentiell bearbeitet, startet der parallele Server neue Prozesse oder Threads um die Aufgabe der Bearbeitung und Beantwortung der Anfrage zu übernehmen und kann selbst wieder auf neue Anfragen lauschen. Im Allgemeinen erlaubt die Verwendung von Threads eine performantere Implementierung als die Verwendung von Prozessen. Threads können hierbei in unterschiedlichen Ebenen, also im Benutzer- oder Systemmodus laufen, wobei beide Methoden Vor- und Nachteile besitzen, die für die jeweilige Implementierung genau abgewogen werden müssen. Als letztes wurde das Konzept des Preforking bzw. Prethreading eingeführt. Hierbei werden Prozesse bzw. Threads bereits im Vorfeld erzeugt um dies nicht erst während der eigentlichen Anfrage des Clients machen zu müssen. Dieses Konzept verbessert somit den parallelen Server.
Grundsätzlich macht die Verwendung von Threads das Programm sehr komplex und ist häufig die Quelle von Fehlern.

4. EREIGNISORIENTIERTE PROGRAMMIERUNG

Das folgende Kapitel führt das ereignisorientierte Programmierparadigma ein. Hierbei wird in Kapitel 4.1 eine Einführung in die Thematik gegeben. Darauf folgend wird Node vorgestellt, ein aktueller Vertreter dieser Sprachen.
Ein weiteres Beispiel für eine starke Orientierung an Ereignisse ist die Programmiersprache Erlang. Sie wurde speziell für die Entwicklung hochperformanter und paralleler Programme entwickelt. Anfangs konnte diese Sprache nur für Telekommunikationsanwendungen eingesetzt werden, da nur dort solche Anforderungen bestanden. Heute allerdings treten genau diese Anforderungen vermehrt auf, weshalb aktuelle Entwicklungen wie CouchDB oder auch ejabberd, worauf der Facebook-chat basiert, mit Erlang realisiert wurden.

4.1 Einführung

Im täglichen Leben treten Ereignisse sehr häufig auf, wie das Klingeln eines Telefons oder das Herunterfallen eines Schlüssels. Auf manche Ereignisse reagieren wir, während andere für uns uninteressant sind. Auch gibt es unerwartete Ereignisse wie einen Banküberfall oder auch ein Lottogewinn. Einige dieser Ereignisse sind sehr einfach zu beobachten, wie beispielsweise Dinge, die wir hören oder sehen, während andere nur durch vorhergehende Aktionen erkannt werden. So könnte das Ereignis „Aktien verkaufen“ auf das Lesen der Zeitung folgen.
In der Informationstechnik ist das Konzept der Ereignisse nicht neu. Sie treten beispielsweise in Form von Exceptions auf, die ein Programm unterbrechen um das Auftreten eines Ereignisses zu zeigen. Ein klassisches Beispiel wäre hier die Berechnung einer Division durch die Zahl Null. Ein weiteres Beispiel sind Benutzeroberflächen wie Java AWT, wo Mausklicks Ereignisse erzeugen und UI-Elemente wie Buttons darauf reagieren.
Obwohl Ereignisse etwas sehr natürliches sind und sie in der realen Welt sehr häufig anzutreffen sind, orientieren sich klassische Programmierparadigmen eher an synchronen Interaktionen, wie dem Anfrage-Antwort Muster. Hierbei führt ein Programm genau das auf Anfrage aus, wofür es geschrieben ist. Der ereignisorientierte Ansatz verfolgt hingegen das Ziel, dass Programme Ereignisse erkennen und selbständig entscheiden ob und wie sie darauf reagieren.[3].

4.2 Ereignisorientierte Programmierung

Verteilte Anwendungen, basieren häufig auf dem Anfrage-Antwort Interaktionsmuster. Dieses Muster ist sehr stark an die Client-Server-Architektur angelehnt, wobei der Server der Diensterbringer und der Client der Dienstnutzer ist, das heißt der Client stellt eine Anfrage und der Server beantwortet sie. Dieser Ablauf ist in Abbildung 3 in Form eines UML-Sequenzdiagramms dargestellt. Beispiele dieses Interaktionsmusters sind *REST*- und *SOAP-webservices* oder auch *remote procedure calls*. Eine Anfrage kann unterschied-

doi: 10.2313/NET-2012-04-1_3

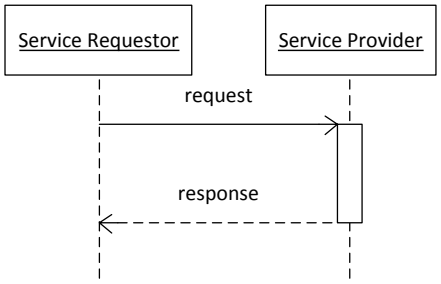


Abbildung 3: Anfrage Antwort Interaktionsmuster (vgl. [3])

licher Natur sein. So ist sie häufig eine Anfrage von Daten oder dem Ergebnis einer Berechnung wie *addiere drei und fünf und gibt das Ergebnis zurück* oder sie besitzt einen *update* Charakter wie *erhöhe die Menge an Bleistiften im Lager um 500*. Die dazugehörige Antwort gibt dann im ersten Beispiel die Zahl acht zurück während im zweiten Beispiel eine Antwort wie *Menge erfolgreich erhöht* sinnvoll ist. Vorteil von diesem Konzept ist, dass jeder Programmierer bereits damit vertraut ist. Außerdem passt dieses Interaktionsmuster häufig sehr gut in das Design von Applikationen. Trotzdem kommen Entwickler häufig an den Punkt, an dem eine Antwort nicht sofort geschickt werden kann, weil der Service Provider noch mit der Auswertung der Anfrage beschäftigt ist. Ein Beispiel hierfür wäre die Anfrage: *„Besitzt diese Instanz des Postschen Korrespondenzproblems eine Lösung? “*. Hierbei kann der Dienstanbieter eine Lösung liefern oder unendlich lange damit beschäftigt sein, folglich kann der Dienstnutzer im Zweifel unendlich lange auf die Antwort warten. Diese Form der Interaktion wird synchron genannt. Die meisten Anfrage-Antwort Interaktionen laufen synchron ab, das heißt der Sender ist so lange blockiert, bis er eine Antwort erhalten hat. Dieses Problem wird häufig mit *Timeouts* behandelt, das heißt es wird nur eine gewisse Zeit auf die Antwort gewartet. Danach wird die Anfrage abgebrochen. Die Verwendung von Ereignissen, kann hierbei aber sehr viel eleganter sein.

Ein Event bezeichnet etwas, was bereits passiert ist, während eine Anfrage eher die Aufforderung darstellt, dass etwas passieren soll. Wegen diesem Unterschied wird im Kontext von Ereignisse nicht von Dienstanbieter und Dienstnutzer, sondern von Ereignisproduzent und Ereigniskonsument gesprochen. Neben dem eben genannten, existieren noch weitere Unterschiede zwischen dem Anfrage-Antwort Muster und der eventgetriebenen Interaktion. Ein Ereignisproduzent, der ein Event an einen Konsumenten geschickt hat, erwartet nicht, dass dieser eine Aktion daraufhin durchführt. Sendet beispielsweise ein Sensor in einem Druckkessel laufend Ereignisse der Art „Mein gemessener Druck ist innerhalb der Grenzwerte“ so muss die Regelung nicht darauf reagieren. Des weiteren werden Ereignisse oftmals als einfache Nachricht gesendet, ohne dass eine Antwort des oder der Konsumenten erwartet wird. Dies bedeutet auch, dass der Ereignisproduzent nach dem Senden des Ereignisses sofort mit seinem Ablauf fortfahren kann, ohne auf eine Antwort zu warten. Dies wird *asynchrone Kommunikation* genannt. Abbildung 4 zeigt eine solche Interaktion. Hier ist zu sehen, dass der Ereignisproduzent die Kommunikation initi-

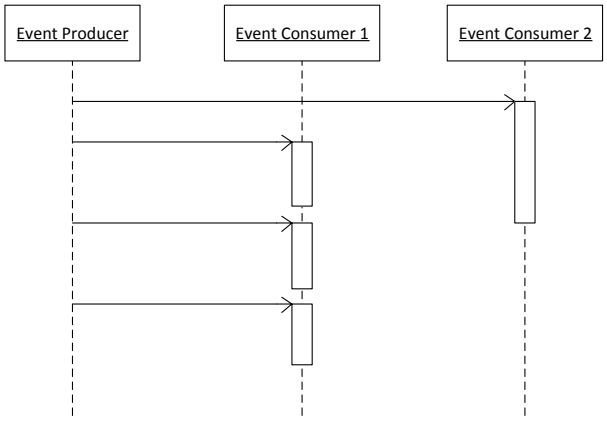


Abbildung 4: Eventtypische Push Interaktion (vgl. [3])

iert, sobald er ein Event zu verschicken hat. Man nennt dieses Vorgehen *Push*. Die Nachrichten die er versendet, sind Einwegnachrichten, es gibt also keine Antworten auf die er warten müsste. Diese Push-Benachrichtigung kann zu einer verbesserten Performance führen.
Um solche Push-Benachrichtigungen durchzuführen, muss der Ereignisproduzent über die Anzahl der Konsumenten und deren Adressierung informiert sein. Dies kann über die folgenden Mechanismen realisiert werden:

- Sämtliche, notwendige Informationen über die Konsumenten können fest im Ereignisproduzenten konfiguriert sein
- Der Ereignisproduzent findet die Konsumenten über einen Directory Service
- Die Ereigniskonsumenten müssen sich beim Produzenten registrieren, bevor sie Ereignisse erhalten

Jede dieser Methoden zieht natürlich eine höhere Belastung für den Ereignisproduzenten nach sich. Hierbei kann auch eine Middleware eingesetzt werden, die die Konsumenten verwaltet, einen sogenannten *Eventchannel*[3].
Die Konzepte der eventorientierten Programmierung gehen natürlich weit über das bisher eingeführte hinaus. Das bisherige reicht jedoch aus, um die Vorteile der Nutzung von Ereignisse bezüglich Client-Server Architekturen darzustellen.

4.3 Node

Node ist eine sehr neue, hochperformante Laufzeitumgebung für JavaScript. Im folgenden Kapitel wird diese genauer untersucht. Nach einer Einführung wird auf das Konzept des *Event-Loops* eingegangen und warum dieses so effizient ist. Hierbei wird eine beispielhafte TCP-Serverimplementierung mit Node genauer erläutert.

4.3.1 Einführung

Node ist eine JavaScript Laufzeitumgebung die es ermöglicht, JavaScript Programme außerhalb von Webbrowsern zu starten. Hierbei bietet es nicht nur die JavaScript-typische

doi: 10.2313/NET-2012-04-1_3

Funktionalität, sonder weitaus mehr, wie das verarbeiten von binären Datenströmen oder das Implementieren verschiedener Server. Node basiert auf der Google V8 Laufzeitumgebung die sehr performant, aber eben auf die JavaScript Funktionalität beschränkt ist. Deshalb wird diese mittels neuer APIs um POSIX Funktionen erweitert. Node nutzt eine Architektur Namens *Event-loop*, die in Kapitel 4.3.2 vorgestellt wird.
JavaScript selbst wurde im Jahr 1995 als eine einfache Scriptsprache zur Nutzung innerhalb von Webseiten für den Netscape Browser entwickelt. Als Webseiten immer dynamischer und schneller wurden und der Bedarf an Funktionalität innerhalb der Webseiten stieg wurde der Begriff *AJAX* populär, wobei das „J"in AJAX für JavaScript steht. Hierbei gab es auch keine wirkliche Auswahl, JavaScript war die einzige auf allen gängigen Browser verfügbare Programmiersprache. In diesem Zuge wandelte sich das Ansehen von Javascript von einer Sprache, die gerade gut ist um Webformulare zu überprüfen hin zu einer vollwertigen Programmiersprache. In letzter Zeit wurde darauf aufbauend etliche professionelle Frameworks entwickelt, wie jQuery, Dojo oder Prototype, die die Programmierung mit JavaScript erleichtern. Nahezu jeder Web-Programmierer hat schon einmal mit JavaScript gearbeitet, das heißt die Verbreitung und der Bekanntheitsgrad ist sehr hoch. Im Browser Wettstreit wurden darüber hinaus immer effizientere JavaScript Laufzeitumgebungen wie die V8 entwickelt, was nicht zuletzt ein Grund war, JavaScript auch auf die Serverseite zu portieren. Zu guter letzt ist JavaScript selbst bereits ereignisorientiert, ein weiterer Grund für die Entwicklung von Node[5].

4.3.2 Event-Loop

Ein grundlegender Teil der Node Architektur ist die *Eventloop*. Im Gegensatz zu vielen anderen Sprachen, bei denen die Nutzung von Ereignisse im Nachhinein dazuentwickelt wurde, ist sie in JavaScript bereits von Anfang an ein Kernkonzept. Dies liegt an der eigentlichen Aufgabe von JavaScript mit Benutzern zu interagieren. Jeder der bereits HMTL-Formulare entwickelt hat, kennt Ereignisse wie *onmouseover* oder *onklick*. Die *Event-loop* ist hierbei ein Konzept, um recht einfach mit solchen Ereignisse zu arbeiten. Dieses Konzept wurde nun von Node aufgegriffen und erweitert. Läuft JavaScript nämlich innerhalb einer Browsers, so gibt es nur eine beschränkte Anzahl an möglichen Ereignisse, wird es allerdings auf den Server portiert, wie es bei Node der Fall ist, kommt eine große Anzahl an Ereignisse hinzu, wie das Ankommen einer Anfrage an einen TCP-server oder das Ergebnis einer Datenbankabfrage.
Die Grundidee bei Node ist, dass alle I/O Aktivitäten nichtblockierend sind. Was bedeutet, dass beim Warten eines TCP-Servers auf eingehende Verbindungen, oder beim Zugriff auf eine Datenbank oder eine einfache Textdatei, das Programm nicht hält und wartet bis ein Ergebnis vorliegt, sondern sofort weiterläuft. Wenn die Daten bereit sind, sprich eine TCP-Anfrage bearbeitet ist oder die Datenbank die Ergebnisse der Abfrage bereit hält wird ein Event erzeugt[5]. Intern wird dies mit den in Kapitel 2.1 vorgestellten select, poll und epoll Funktionen realisiert. Dies bedeutet, dass der Betriebssystemkernel die Anwendung informiert, sobald die Daten bereit sind[10]. Programmiertechnisch ist dies eine Form der asynchronen Kommunikation (vgl. Kapitel 4.2) und wird mit sogenannten callback-Funktionen realisiert. So wird beispielsweise beim Erzeugen eines TCP-Servers, gleich eine Funktion mitübergeben, die aufgerufen wird, sobald ein neue Anfrage ankommt. Diese Funktion heißt callback Funktion. Sie reagiert somit auf die Ereignisse des Betriebssystems. Ein Beispiel hierzu ist in Kapitel 4.3.3 erläutert. Der prinzipielle Ablauf eines Node-Programms zeigt hierbei die Funktionsweise der *Event-loop* recht einfach. Zuerst läuft das Programm komplett von Anfang bis Ende ab. Dies kann als eine Art von Initialisierung aufgefasst werden. Hierbei werden sämtliche *Event-Listener*, sprich die callback-Funktionen registriert. Beinhaltet das Programm keine *Event-Listener* beendet sich das Programm, sonst allerdings bleibt es aktiv bzw. wird vom Betriebssystem schlafen gelegt, wartet auf eingehende Ereignisse und ruft anschließend die jeweiligen Funktionen auf.
Ein Kritikpunkt an Node ist, dass es *single-threaded* ist. Das bedeutet, dass das komplette Programm in einem einzige Thread abläuft. Somit werden die Vorzüge von Mehrprozessor- oder Multithreadingsystemen nicht genutzt. Prinzipiell wäre eine Erweiterung möglich, die das Erstellen mehrerer Threads erlauben, jedoch steigt hiermit die Komplexität des kompletten Systems und somit auch die für den Programmierer. Da Node jedoch niemals auf I/O wartet, ist die Zeit die zwischen dem Ankommen eines Ereignisse und dessen Bearbeitung stets sehr gering[5].

4.3.3 Beispiel: TCP-Server

Um die Funktionsweise von Node und der *Event-loop* zu verdeutlichen, ist in Abbildung 5 eine beispielhafte Implementierung eines TCP-Echo-Servers dargestellt. Er antwor-

```
var net = require('net');

var server = net.createServer(function (socket) {
  socket.write("Echo server\r\n");
  socket.pipe(socket);
});

server.listen(1337, "127.0.0.1");
```

Abbildung 5: Beispielhafte TCP-Server Implementierung mit Node.js

tet auf jede einkommende Anfrage schlichtweg mit dem vom Client eingegebenen Text. In Zeile eins wird hierzu das Modul *net* geladen, welches die Implementierungen eines TCP-Servers enthält. Danach wird mit *createServer()* ein neuer Server erzeugt. Hierbei wird als Funktionsparameter die callback Funktion übergeben. In diesem Beispiel handelt es sich um eine anonyme Funktion. Beim Aufruf der callback Funktion wird wiederum ein Parameter übergeben. Dies ist der Socket, mit dem der Client verbunden ist. Die Funktion selbst schreibt beim Aufruf nun einfach den String „Echo server" und den vom Client eingegebenen Text auf den Socket. Die letzte Zeile des Beispielcodes veranlasst den Server mit der lokalen IP Adresse auf dem Port 1337 zu lauschen. Nachdem das Programm nun einmal durchgelaufen ist, sozusagen die Initialisierung beendet hat, wird der Thread vom Betriebssystem schlafen gelegt. Er verbraucht somit keine Rechenleistung. Meldet das Betriebssystem nun das Ereignis einer neue Anfrage, reagiert das Programm mit dem Aufruf der callback Funktion, antwortet also mit „Echo server" und

doi: 10.2313/NET-2012-04-1_3

dem jeweiligen Text und wird anschließend vom Betriebssystem wieder schlafen gelegt.

4.4 Zusammenfassung

Ereignisse begegnen einem im Alltag sehr häufig, weshalb sie sehr natürlich sind. Im Gegensatz zum Anfrage-Antwort Interaktionsmuster, werden Ereignisse als Einwegnachrichten versendet. Der Ereignisproduzent erwartet also keine Antwort. Dies wird auch als *Push* bezeichnet. Der Ereigniskonsument entscheidet selbst, ob und mit welcher Aktion er auf ein Event reagiert, da nicht alle Ereignisse eine Reaktion erfordern.
Node ist eine neue Laufzeitumgebung für JavaScript, die auf Ereignissen basiert. Sie nutzt hierbei das Konzept der Event-loop, die nach einer ersten Registrierung sämtlicher Ereignisse nur noch auf eingehende Ereignisse wartet, wobei das Programm selbst schlafen gelegt wird. Kommt ein solches Ereignis an, wird eine speziell dafür registrierte callback Funktion aufgerufen. Diese reagiert somit auf eintreffende Ereignisse. Durch dieses Konzept, ist sämtliche I/O Operation in Node nicht blockierend, was enorme Performancesteigerungen nach sich zieht. Hierbei wird ein Node Programm nur in einem Thread ausgeführt, was ein häufiger Kritikpunkt an diesem Konzept ist.

5. PERFORMANCEVERGLEICH

In diesem Kapitel werden einige der vorgestellten Konzepte verglichen. Hierzu wurden verschiedene TCP Server in Python und Node umgesetzt deren Quellcodes in Kapitel A dargestellt sind. Die Server liefen hierbei auf einem Multicore Atom Rechner mit dem Betriebssystem FreeBSD 8.2. Die Anfragen wurden von einem Linux Rechner (Ubuntu 11.10) über ein Gigbit Ethernet gesendet. Zur Messung der Zeiten wurde das Apache Benchmark Tool in der Version 2.3 Revision 655654 eingesetzt wobei jeder Test mehrmals durchgeführt wurde. Die im Folgenden präsentierten Werte entsprechen den Mittelwerten dieser Durchläufe.

5.1 Einfacher HelloWorld-Server

Der erste Test wurde mit einfachen HelloWorld-Servern durchgeführt, welche schlicht den String *HelloWorld* zurück geben. Hierbei wurden eintausend Anfragen an den Server gesendet, wobei zehn Verbindungen gleichzeitig genutzt wurden. Tabelle 1 zeigt die entsprechenden Messergebnisse. In der ersten Spalte die Gesamtzeit des Tests, in der zweiten die Anzahl an Antworten des Servers pro Sekunde.

	t [s]	req/t [1/s]
Iterativer Server	1.1148	897.928
Paralleler Server - Prozesse	4.4712	220.562
Paralleler Server - Threads	1.8856	530.664
Node Server	1.3624	743.41

Tabelle 1: Performancevergleich, „HelloWorld“ Server

Auffällig ist, dass der iterative Server am schnellsten ist, da hier der gesamte Test in der kürzesten Zeit ablief und dieser Server am meisten Antworten pro Sekunde schicken konnte. Dieser Test ist dahingehend realitätsfern, da er schlicht einen kurzen String zuzrückliefert, was eher selten anzutreffen ist. Es werden serverseitig also keinerlei Berechnungen oder Zugriffe auf andere Ressourcen durchgeführt. Deshalb ist es auch nicht verwunderlich, dass der iterative Server hier am schnellsten abschneidet. Er muss weder neue Threads oder Prozesse kreieren, noch auf Events reagieren oder callback Funktionen aufrufen. Bei den parallelen Servern ist der mit Theads realisierte Server deutlich schneller als der mit Prozessen. Hierbei ist der Performancevorteil von Threads gegenüber Prozessen deutlich zu sehen. Der Server, der mit Node realisiert wurde, ist zwar langsamer als der iterative, jedoch den parallelen Servern deutlich voraus.

5.2 Berechnungen auf der Serverseite

Ein realitätsnahes Szenario wurde in diesem Test untersucht. Hierbei wird auf der Serverseite bei jeder Anfrage hundert Millisekunden gewartet bis die Antwort geschickt wird. Dies soll eine Aktion simulieren, wie beispielsweise eine Berechnung, einen Datenbankzugriff oder den Zugriff auf externe Ressourcen. Dieser Test wurde mit insgesamt hundert Anfragen durchgeführt, mit jeweils fünf parallelen Verbindungen. Tabelle 2 zeigt die entsprechenden Ergebnisse. Wie beim vorherigen Test die Gesamtzeit, und die Antworten pro Sekunde. Darüber hinaus ist hier Zeit pro Anfrage dargestellt.

	t [s]	req/t [1/s]	t/req [ms]
It. S.	10.461	9.56	104.606
P. S. - Prozesse	2.443	40.94	122.146
P. S. - Threads	2.306	43.373	115.288
Node S.	2.251	44.443	112.522

Tabelle 2: Performancevergleich, Server mit Berechnung

Die Gesamtzeit des Tests des iterativen Servers liegt bei knapp über zehn Sekunden, was nicht verwunderlich ist, da alle hundert Anfragen nacheinander abgearbeitet werden, wobei jede einhundert Millisekunden dauert. Auch für die parallelen und den Node Server ist die Gesamtzeit interessant. Der Optimalwert wäre hier zwei Sekunden, da je fünf Anfragen parallel geschickt wurden. Wie an den Ergebnissen zu sehen ist, ist der Node Server diesem Wert am nächsten. Auch hier ist der parallele Server mit Threads schneller als der mit Prozessen. Der Unterschied der drei letztgenannten Server ist in diesem Beispiel gering. Er steigt jedoch mit der Anzahl an parallelen Verbindungen, da genau hier die Stärken der parallelen Server dem iterativen bzw. des Node-Servers den parallelen gegenüber liegt.

6. FAZIT

Zum Entwurf hochperformanter Client-Server Architekturen bedarf es vor allem an Kenntnissen in den Programmiergrundlagen und den I/O Modellen. Diese wirken sich stark auf die Effizienz des Servers aus. So können je nach gewähltem Modell Prozesse vollständig blockieren, während sie auf I/O warten, oder auch benachrichtigt werden sobald die jeweiligen Ressourcen verfügbar sind.
Die Art der Serverimplementierungen ist eine weitere wichtige Eigenschaft effizienter Realisierungen. Hierbei erwies sich der parallele Server als eine Möglichkeit, Interaktion mit vielen Clients schneller zu ermöglichen, als die Anfragen sequentiell zu bearbeiten. Die Nutzung von Threads, die leichtgewichtiger als Prozesse sind erwies sich als eine Möglichkeit, den parallelen Server performant zu implementieren. In Kapitel 5 wurde deutlich, dass bei mehreren Clients

doi: 10.2313/NET-2012-04-1_3

der parallele Server deutlich schneller ist als der iterative. Eine weitere Verbesserung bietet dabei die Ausnutzung von Prethreading bzw. Preforking, um die Erzeugung der Prozesse bzw. Threads nicht auf Kosten der Antwortzeit durchzuführen.
Die Nutzung von Threads birgt jedoch auch Nachteile. So werden Programme, die diese Nutzen, sehr komplex und das Debuggen und Testen gestaltet sich als äußerst schwierig. Des weiteren ist eine sorgfältige Wahl des Threadmodells notwendig. Die Kosten der Erzeugung von Threads ist ein weiterer Nachteil.
Ein aktueller Ansatz, der die komplizierte und aufwendige Nutzung von Threads vermeidet, ist der eventgetriebene. Als ein Vertreter dieses Paradigma wurde Node vorgestellt. Es folgt strikt dem Ansatz, sämtliche I/O als nichtblockierend zu realisieren. Dabei werden I/O Operationen mit callback Funktionen ausgestattet, die aufgerufen werden, sobald die Operationen selbst beendet sind. Dadurch muss das eigentliche Programm, niemals auf Daten warten. Dies bedeutet auch, dass weder Prozesse noch Threads erzeugt werden müssen. Dieser Vorteil wurde auch in Kapitel 5 deutlich, indem eine Serverimplementierung in Node in den meisten Fällen den anderen Konzepten überlegen war.

7. LITERATUR

[1] G. Bengel: *Grundkurs verteilte Systeme*, S. 37-44, Vieweg+Teubner Verlag, 2004

[2] J. Corbet, A. Rubini, G. Kroah-Hartman: *Linux device drivers*, S. 163-165, O'Reilly Media, Inc., 2005

[3] O. Etzion, P. Niblett: *Event Processing in Action*, S. 3-58, Manning Publications Co., 2011

[4] E. Glatz: *Betriebssysteme - Grundlagen, Konzepte, Systemprogrammierung*, S. 91-98, 293-301, dpunkt.verlag., Heidelberg, 2006

[5] *Preview of Node: Up and Running: Scalable Server-Side Code with JavaScript* , `http://ofps.oreilly.com/titles/9781449398583/index.html`

[6] *The C10K problem* , `http://www.kegel.com/c10k.html`

[7] R. W. Stevens: *Programmieren von UNIX-Netzwerken*, S. 139-146, 719-737Carl Hanser Verlag, München, Wien, 2000

[8] *Advantages and Disadvantages of a Multithreaded/Multicontexted Application*, `http://download.oracle.com/docs/cd/E13203_01/tuxedo/tux71/html/pgthr5.htm`

[9] J. Wolf: *Linux-UNIX-Programmierung*, S. 255-257, 367-371, Galileo Press, 2009

[10] *The Node.js Website*, `http://nodejs.org/l`

[11] J. Armstrong: *A History of Erlang*, Ericcson AB, S. 1, 16-20, ACM New York, NY, USA, 2007

[12] A. Tanenbaum: *Moderne Betriebssysteme, 3. Auflage*, S. 137, Pearson Studium, München, 2009

ANHANG

A. PROGRAMM-QUELLCODES

A.1 Iterativer TCP Server in Python

```
import SocketServer
import time

class MyTCPHandler(SocketServer.BaseRequestHandler):
  def handle(self):
    time.sleep(0.1)
    self.request.send("HelloWorld\n")

server = SocketServer.TCPServer(("192.168.50.5", 9999),
   MyTCPHandler)
server.serve_forever()
```

A.2 Paralleler TCP Server in Python mit Threads

```
import socket
import threading
import SocketServer
import time

class ThreadedTCPRequestHandler(
   SocketServer.BaseRequestHandler):

  def handle(self):
    time.sleep(0.1)
    self.request.send("HelloWorld\n")

class ThreadedTCPServer(SocketServer.ThreadingMixIn,
   SocketServer.TCPServer):
  pass

server = ThreadedTCPServer(("192.168.50.5", 9997),
   ThreadedTCPRequestHandler)

server_thread = threading.Thread(
   target=server.serve_forever)
server_thread.daemon = False
server_thread.start()
```

A.3 Paralleler TCP Server in Python mit Prozessen

```
import os
import SocketServer
import time

class ForkingTCPRequestHandler(
   SocketServer.BaseRequestHandler):

  def handle(self):
    time.sleep(0.1)
    self.request.send("HelloWorld\n")

class ForkingTCPServer(SocketServer.ForkingMixIn,
   SocketServer.TCPServer):
  pass

import socket
import threading

server = ForkingTCPServer(("192.168.50.5", 9998),
   ForkingTCPRequestHandler)

server_thread = threading.Thread(
   target=server.serve_forever)
server_thread.daemon = False
server_thread.start()
```

A.4 Node TCP Server

```
var net = require('net');

var server = net.createServer(function (socket) {
  setTimeout(function () {
    socket.end("HelloWorld\n");
  }, 100)
});

server.listen(9996, "192.168.50.5");
```


doi: 10.2313/NET-2012-04-1_3

Privacy and Smart Meters / Smart Grid

Thomas Oberwallner
Betreuer: Dr. Heiko Niedermayer
Hauptseminar - Innovative Internettechnologien und Mobilkommunikation WS 2011/2012
Lehrstuhl Netzarchitekturen und Netzdienste
Fakultät für Informatik, Technische Universität München
Email: thomas.oberwallner@mytum.de

KURZFASSUNG

Durch die zunehmende dezentrale Stromversorgung durch erneuerbare Energien muss das Stromnetz den Strom über weitere Strecken transportieren. Um eine zuverlässige Versorgung zu erreichen, ist es daher notwendig, dass die Betreiber die Auslastung zuverlässig messen können. In dieser Ausarbeitung wird beschrieben, welche Probleme im Bereich Datenschutz bei der Einführung des Smart Grids (intelligenten Stromnetzes) und Smart Meter (intelligenter Stromzähler) auf die Verbraucher zukommen. Es werden zwei Protokolltypen (grundlegendes Protokoll und No-Leakage Protokoll) dargestellt, die sowohl eine Lastüberwachung des Stromnetzes, als auch eine regelmäßige Rechnungsstellung der Stromverbraucher ermöglichen, ohne die Privatsphäre der Nutzer zu gefährden. Bislang sind diese Protokolle jedoch noch nicht in den Geräten implementiert.

Schlüsselworte

Smart Meter, Smart Grid, Privacy, Protocol, No-Leakage, Datenschutz

1. EINLEITUNG

1.1 Begriffe Smart Grid/ Smart Meter

Der Begriff Smart Grid kommt aus dem Englischen und wird meist mit intelligentem Stromnetz oder Energieinformationsnetz übersetzt. Er bezeichnet die Vernetzung zwischen Stromerzeuger, Stromverbraucher, Stromspeicher und Stromübertrager. Dieser Vernetzung kommt eine hohe Bedeutung zu, da aktuell ein Übergang von zentraler Stromerzeugung durch Großkraftwerke wie Kohle-, Gas- oder Atomkraftwerke hin zu dezentraler Versorgung durch erneuerbare Energien stattfindet. Wenn im südlichen Teil Deutschlands wenig Sonne scheint, muss beispielsweise der aus Windkraft erzeugte Strom von der Nordsee nach Bayern transportiert werden, da die Grundlastkraftwerke im südlichen Teil von Deutschland den Strombedarf eventuell nicht decken können. In Deutschland war im Jahr 2010 eine Photovoltaik-Nennleistung von 17370 Megawatt-Peak installiert [6], was in etwa der Gesamtleistung aller Atomkraftwerke in Deutschland entspricht. Insgesamt wurden jedoch nur etwa 12000 Gigawattstunden Strom [6] produziert, was in etwa der Jahresstromproduktion des Kernkraftwerks Isar 2 entspricht [4]. Also ist die durch Photovoltaik erzeugte Energiemenge stark schwankend und sollte, wenn bei sonnigem Wetter Stromüberfluss vorhanden ist, zum Beispiel in Pumpspeicherkraftwerken gespeichert werden.

Smart Meter bezeichnet intelligente Zähler, die den aktuellen Verbrauch an Strom, Wasser, Gas oder Fernwärme kontinuierlich ermitteln und anzeigen. Zusätzlich wird wie in [8] davon ausgegangen, dass sie zu einer Zwei-Wege-Kommunikation fähig sind, also den Stromverbrauch auch an den Versorger weiterleiten können. In dieser Ausarbeitung beschränke ich mich auf das Einsatzgebiet im Stromnetz, allerdings lässt sich ein Großteil der Schlussfolgerungen auch auf andere Verbraucher übertragen. Ziel dieses Messgeräts ist es, den Stromverbrauch häufiger an den Versorger zu übermitteln, um die Rechnungsstellung anhand des aktuellen Verbrauchs in kürzeren Intervallen zu ermöglichen und damit einerseits tageszeitabhängige Stromtarife anzubieten und andererseits den Kunden für den eigenen Stromverbrauch zu sensibilisieren und somit den Energieverbrauch zu senken. Damit können Kunden zum Beispiel erkennen, wie hoch der Stromverbrauch der nachts auf Standby laufenden Geräte ist. Ebenfalls können Anreize geschaffen werden, damit Kunden den Strom dann beziehen, wenn gerade ausreichende Mengen davon im Netz sind und somit Lastspitzen im Netz des Netzbetreibers zu senken. Für den Kunden haben intelligente Zähler jedoch einige Nachteile: So kosten diese Zähler deutlich mehr als normale Drehstromzähler (der einzelne Nutzer muss jedoch nicht die Anschaffung zahlen, sondern diese wird auf alle Kunden in Form einer Grundgebühr umgelegt) und haben möglicherweise einen nicht zu vernachlässigenden Eigenverbrauch. Ebenfalls werden durch tageszeitabhängige Tarife unflexible Kunden benachteiligt, die nicht die Zeit haben, zu günstigen Tarifen Strom zu verbrauchen und somit mit deutlich höheren Stromkosten rechnen müssen. Zusätzlich ist für die Übermittlung der Verbrauchsdaten eine Internetverbindung nötig, die - falls nicht bereits vorhanden - weitere Kosten verursacht. Eine weitere Gefahr besteht auch darin, dass der Stromversorger nun die Möglichkeit erhalten könnte, die Stromversorgung des Nutzer abzuschalten, falls dieser seine Rechnungen nicht bezahlt.

Das aus Sicht der Informatik interessanteste Problem ist jedoch, dass die Privatsphäre der Kunden in Gefahr ist, wenn die Datenübertragung ungesichert oder in zu kurzen Abständen erfolgt und Stromanbieter somit Details über Lebensgewohnheiten ihrer Kunden erfahren, die nicht öffentlich werden sollten. Diese Gefahren werden im folgenden Kapitel erörtert. Kapitel 2 beschreibt, wie Smart Meter mit den Stromverbrauchern im Haushalt und mit dem Internet verbunden sind. Anschließend wird auf die internationale Umsetzung von Smart Metern eingegangen. In Kapitel 4 werden zuerst die Anforderungen an Übertragungsprotokolle definiert und anschließend Protokolle vorgestellt, die sowohl eine regelmäßige Rechnungsstellung als auch eine Lastermittlung ermöglichen, ohne die Privatsphäre der Kunden

doi: 10.2313/NET-2012-04-1_4

zu gefährden.

1.2 Mögliche Gefahren für die Privatsphäre

Wenn von einer unverschlüsselten Übertragung des aktuellen Stands des Stromzählers ausgegangen wird, hängen die Inferenzmöglichkeiten von der Häufigkeit der Übertragung ab: Wird der Zählerstand täglich übertragen, so kann der Energieversorger (oder mögliche Einbrecher, die die Datenübertragung des Smart Meters abfangen) erkennen, ob der Nutzer an diesem Tag zu Hause war. Falls der Nutzer eventuell mehrere Tage abwesend war, könnte ein Einbrecher darauf schließen, dass das potentielle Opfer im Urlaub ist und wahrscheinlich auch am folgenden Tag nicht zu Hause sein wird. Wenn die Messungen häufiger, also zum Beispiel stündlich oder sogar viertelstündlich erfolgen, kann man die Lebensgewohnheiten der Kunden erkennen. Beispielsweise ist anhand der Verbrauchsdaten ersichtlich, wann ein Nutzer aufsteht, wann er in die Arbeit fährt und wann er wieder nach Hause kommt, oder auch wie viele Personen im Haushalt leben. Letzteres ist insbesondere dann möglich, wenn nicht nur der Strom-, sondern auch Wasser- und Gas-/Fernwärmeverbrauch übertragen wird. Bei sekündlicher Messung können Details über die aktuelle Tätigkeit des Kunden ermittelt werden. So lässt sich beispielsweise durch die unterschiedliche Leistungsaufnahme des Fernsehers in Abhängigkeit der Bildschirmhelligkeit sogar der aktuelle Fernsehsender des Kunden ermitteln [11]. Unabhängig von der genauen Übertragungshäufigkeit (so lange die Übermittlung zumindest täglich stattfindet), sind auch langfristige Analysen des Stromverbrauchs des Kunden möglich [7]. So kann beispielsweise gezielte Werbung geschaltet werden, wenn der Kühlschrank des Kunden älter wird und somit zunehmend ineffizient arbeitet, oder wenn ein Haushalt einen hohen Standby-Verbrauch besitzt.

2. INFRASTRUKTUR IM HAUSHALT

In diesem Kapitel wird kurz auf die Infrastruktur des Smart-Meters im Haushalt eingegangen (siehe Abbildung 1) [3]: Das Gateway stellt die zentrale Kommunikationseinheit der Infrastruktur dar. Es verbindet ein oder mehrere Smart Meter im Lokalen Metrologischen Netz (LMN) mit dem Internet (WAN). Ebenfalls sind Energieverbraucher aus dem Haushaltsnetzwerk (Home Area Network, HAN) angeschlossen. In Mehrfamilienhäusern stellen Gateways Knoten dar, die die Verbrauchdaten der einzelnen Smart Meter regelmäßig verschlüsselt übermittelt bekommen und diese signiert im eigenen Speicher ablegen. Anschließend werden diese Daten vom Gateway an den Stromversorger übermittelt. Zusätzlich muss das Gateway sicherstellen, dass nur berechtigte Zugriffe von außen auf Daten des/der Smart Meters zugelassen werden. Diese Darstellung stellt nur die logische Sicht dar, in Einfamilienhäusern können Smart Meter durchaus auch direkt in das Gateway eingebaut werden. Deswegen wird im Weiteren nicht mehr zwischen den Aufgaben des Gateways und des Smart Meters unterschieden, sondern es wird davon ausgegangen, dass ein Gerät, das „Smart Meter“, die genannten Funktionen beinhaltet.
Zusätzlich ist denkbar, dass wichtige oder sicherheitskritische Stromverbraucher, wie Außenbeleuchtung, Rollos, Herd oder Backofen mit dem Gateway verbunden sind, so dass der Stromkunde diese von der Ferne aus steuern kann. Diese Vernetzung wird als Smart Living oder intelligentes Wohnen bezeichnet. Es wird allerdings in dieser Arbeit nicht genauer darauf eingegangen, da die Steuerung der Hausgeräte im Allgemeinen nicht zu Smart Metering gezählt wird.

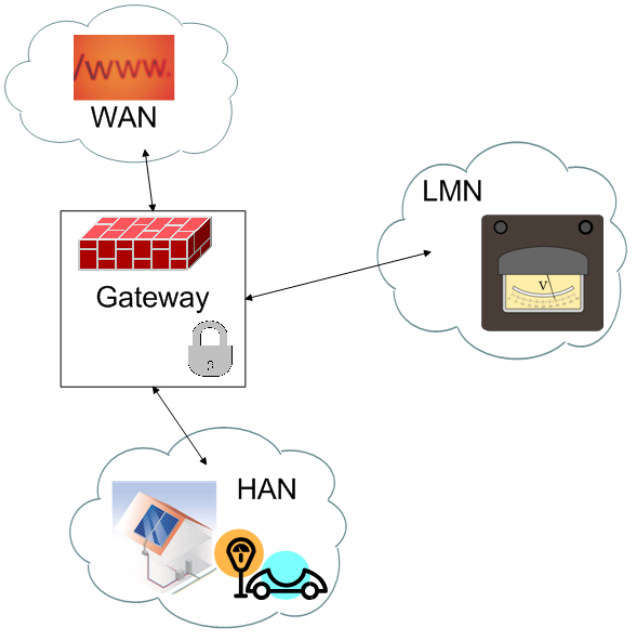


Abbildung 1: Infrastruktur der Smart Meter (LWN), des Gateways, der Verbraucher (HAN) im Haushalt nach [3]

3. INTERNATIONALE UMSETZUNG

3.1 Gesetzliche Vorgaben in der EU/Deutschland

Die Rahmenbedingungen für den Einsatz von intelligenten Stromnetzen beziehungsweise intelligenten Stromzählern sind in der EU-Richtlinie 2006/32/EG festgelegt. Ziel dieser Richtline war es, Anreize zu schaffen, um den Energieverbrauch der Mitgliedsstaaten mittelfristig um 20% zu senken. In Bezug auf intelligente Stromzähler enthält sie folgenden Abschnitt:
„Soweit es technisch machbar, finanziell vertretbar und im Vergleich zu den potenziellen Energieeinsparungen angemessen ist, stellen die Mitgliedstaaten sicher, dass, alle Endkunden in den Bereichen Strom, Erdgas, Fernheizung und/oder -kühlung und Warmbrauchwasser individuelle Zähler zu wettbewerbsorientierten Preisen erhalten, die den tatsächlichen Energieverbrauch des Endkunden und die tatsächliche Nutzungszeit widerspiegeln.“[1]
Ebenfalls ist vorgegeben, dass intelligente Stromzähler für Neubauten und Totalsanierungen verpflichtend sind. Dies wurde in Deutschland im Energiewirtschaftsgesetz (EnWG) umgesetzt. So ist der Netzbetreiber hierzulande seit Anfang 2010 für den Einbau einer „Mindestlösung“ zuständig, in der keine Funktion der Fernübertragung des aktuellen Stromverbrauchs eingebaut ist. Dieses Messgerät kann somit nur den Stromverbrauch aufzeichnen und ermöglicht dem Stromkunden einen Überblick über den eigenen Stromverbrauch. Der Entwurf der Europäischen Kommission für eine neue

[1]RICHTLINIE 2006/32/EG DES EUROPÄISCHEN PARLAMENTS UND DES RATES vom 5. April 2006 über Endenergieeffizienz und Energiedienstleistungen und zur Aufhebung der Richtlinie 93/76/EWG des Rates

doi: 10.2313/NET-2012-04-1_4

Energieeffizienzrichtlinie sieht vor, dass spätestens bis zum 01.01.2015 der Stromverbrauch monatlich nach tatsächlichem Verbrauch abgerechnet wird[2].
Für den Datenschutz ist die EG-Datenschutzrichtlinie 95/46/EG anwendbar, die Vorgaben hinsichtlich der Verarbeitung personenbezogener Daten festlegt. Genauere Vorgaben in Bezug auf die Nutzung personenbezogener Daten im intelligenten Stromnetz sind im Energiewirtschaftsgesetz vorhanden[3]: So dürfen unter anderem nur personenbezogene Daten von berechtigten Stellen erhoben, verarbeitet und genutzt werden, um beispielsweise den Energieverbrauch und die Einspeisemenge zu messen und abzurechnen, oder um variable Tarife umzusetzen. Diese Stellen müssen Anforderungen aus §4a des Bundesdatenschutzgesetzes genügen.

3.2 Verlauf in den Niederlanden

In den Niederlanden schlug die Regierung im Jahr 2007 vor, dass alle Haushalte bis 2013 einen intelligenten Stromzähler erhalten müssen. Die niederländische Wirtschaftsministerin plante sogar, alle Hausbesitzer, die keinen solchen Zähler einbauen wollen, mit einer Geldstrafe von 17.000 Euro oder einer sechsmonatigen Gefängnisstrafe zu belegen [2]. Im weiteren Verlauf traten Verzögerungen auf, weil diese Messgeräte zum damaligen Zeitpunkt keine ausreichenden Möglichkeiten boten, eigene Stromproduktion (zum Beispiel durch Photovoltaik) abzurechnen. Im Jahr 2009 wurde - hauptsächlich wegen möglicher Gefahren für die Privatsphäre - die Einführung von intelligenten Stromzählern nur noch auf freiwilliger Basis beschlossen.

3.3 Aktuelle Smart-Metering-Tarife in Deutschland

In Deutschland bieten aktuell mehrere Energieversorger Tarife mit Smart-Metern an. In diesem Kapitel wird auf den aktuellen Stand der drei größten Energieversorger E.ON, RWE und EnBW eingegangen: Bei E.ON wird derzeit der Tarif E.ON EnergieNavi beworben, bei dem Nacht- und Tagstrom zu unterschiedlichen Preisen verkauft wird. Ebenfalls erhalten Kunden die Möglichkeit, den eigenen Stromverbrauch über ein Webportal in einer Auflösung von bis zu fünfzehn Minuten zu betrachten [5]. Bei RWE wird aktuell kein solcher Tarif angeboten, aber die Möglichkeiten von Smart Metering im Rahmen eines Pilotprojekts erforscht, bei dem eine Fernabfrage des Zählerstands durch den Versorger vorgesehen ist [12]. EnBW bietet ebenfalls einen intelligenten Stromzähler in Verbindung mit einem Tarif an, der nachts und am Wochenende vergünstigte Preise für verbrauchten Strom bietet. Die Kunden können sich den eigenen Stromverbrauch ebenso über ein Webportal ansehen [1]. Somit besteht bei allen drei Versorgern grundsätzlich das Risiko, dass mögliche Angreifer den Stromverbrauch des Kunden detailliert erfahren. Weniger komfortabel, aber deutlich sicherer wäre es, wenn nur der für Abrechnungszwecke benötigte Energieverbrauch an den Versorger übermittelt wird und die Analyse des eigenen Stromverbrauchs nur aus dem eigenen Netzwerk heraus möglich ist. Daten zur Lastermittlung des Stromnetzes sollten nur in anonymisierter Form durch sichere Protokolle übertragen werden. Die Anforderungen an derartige Übertragungsprotokolle werden im folgenden Kapitel erläutert.

[2]RICHTLINIE DES EUROPÄISCHEN PARLAMENTS UND DES RATES zur Energieeffizienz vom 22.06.2011
[3]Energiewirtschaftsgesetz (EnWG) vom 7. Juli 2005, §21g

Tabelle 1: Notation der Protokolle

Notation	Bedeutung
$A \rightarrow B : m$	Nachricht von A an B mit Inhalt m
$\{m\}_A$	Nachricht m von A verschlüsselt
$[m]_A$	Nachricht m von A signiert

4. PROTOKOLLE

4.1 Anforderungen an den Datenschutz

Die Anforderungen an den Datenschutz bei der Erhebung von Smart Meter Daten zu Lastüberwachungszwecken sind in [9] zusammengefasst: Bei der Übertragung der Daten an Energieversorger oder Netzbetreiber muss die Anonymität des einzelnen Kunden gewahrt bleiben, so dass keine Aussagen über die Lebensgewohnheiten des Verbrauchers möglich sind, jedoch müssen Abrechnungsdaten eindeutig einem Smart Meter zugeordnet werden können. Ebenfalls soll die Unverknüpfbarkeit von Datenpaketen gewährleistet sein, damit Pakete des gleichen Smart Meter, die zu unterschiedlichen Zeitpunkten gesendet werden, nicht miteinander verknüpft werden können. Es soll somit bei der Datenübertragung zur Lastübermittlung nicht möglich sein, auf den Verbrauch des einzelnen Kunden zu schließen. Die Daten müssen zusätzlich authentisch sein. Der Netzbetreiber muss erkennen, ob Verbrauchsdaten von nicht-registrierten Smart Metern stammen. Das Entfernen von Smart Metern (und damit das Abweisen von zukünftigen Datenpaketen) muss einfach möglich sein. Spamming- und Replay-Angriffe[4] sind zu verhindern und die Daten müssen weitgehend verzögerungsfrei (beziehungsweise im Zeitraum von wenigen Minuten) übertragen werden können. Wenn möglich, soll auf keine dritte Partei (Trusted Third Party) zurückgegriffen werden, da dies einen höheren Aufwand erfordern würde und eventuell negative Auswirkungen auf die Sicherheit hätte.
Für Abrechnungszwecke sollten grundsätzlich nur die Daten übertragen werden, die der Energieversorger für die Rechnungsstellung benötigt. Wenn das Abrechnungsintervall groß genug (also zum Beispiel monatlich) gewählt ist, bestehen - bei verschlüsselter und signierter Datenübertragung - keine Gefahren für die Privatsphäre.
Im Anschluss werden zwei Protokolle in Anlehnung an [7] vorgestellt, die die Übertragung von Informationen zu Abrechnungs-, aber auch zu Laststeuerungszwecken ermöglichen, jedoch beide noch nicht in aktuellen Messgeräten verwendet werden. Das grundlegende Protokoll berücksichtigt dabei nicht den Datenschutz gegenüber dem Versorger, sondern nur die Authentizität der übermittelten Daten und ist damit ausschließlich zu Abrechnungszwecken geeignet. Die Notation der Protokolle ist in Tabelle 1 dargestellt.

4.2 Grundlegendes Protokoll

Am grundlegenden Protokoll sind drei Parteien beteiligt:

- Netzbetreiber GO

[4]Bei einem Replay-Angriff spielt ein Angreifer einmal abgefangene Daten zu einem späteren Zeitpunkt erneut ein, um somit eine fremde Identität vorzutäuschen.

doi: 10.2313/NET-2012-04-1_4

- Versorger S
- Smart-Meter M

Vor Beginn des eigentlichen Protokolls besitzt das Smart-Meter bereits den öffentlichen Schlüssel des Netzbetreibers (GO) und den öffentlichen Schlüssel der Zertifizierungsstelle. Der Netzbetreiber besitzt den öffentlichen Schlüssel des Smart-Meters. Das grundlegende Protokoll besteht insgesamt aus drei Unterprotokollen. In set_supplier wird dem Smart Meter der zuständige Versorger mitgeteilt, switch_power ermöglicht dem Netzbetreiber die Stromversorgung des Kunden einzuschränken und meter_report sendet den aktuellen Zählerstand des Smart Meters an den Stromversorger. In set_supplier wird dem Smart-Meter vom Netzbetreiber mitgeteilt, wer der zuständige Versorger (S) ist. Zusätzlich wird dem Smart-Meter der öffentliche Schlüssel des Versorgers übermittelt (pk_S), der Zeitpunkt (ts) festgelegt, zu dem der neue Versorger aktiv wird und das Abrechnungsintervall mitgesendet (P):

1. GO → M: hi, init set_supplier
2. M → GO: nonce n
3. GO → M: $\{[\text{set\_supplier, M, n, S, } pk_S\text{, ts, P}]_{GO}\}_M$

Das Smart-Meter kann nun die Nachricht mit dem eigenen privaten Schlüssel entschlüsseln und die Signatur der Nachricht mit dem öffentlichen Schlüssel des Netzbetreibers und die Zufallszahl n verifizieren. Falls dies erfolgreich ist, wird der neue Versorger eingetragen.

Wenn der Netzbetreiber (GO) Wartungsarbeiten am Netz durchführt oder der Kunde seine Rechnungen nicht bezahlt, wird das switch_power Protokoll ausgeführt, das es dem Netzbetreiber ermöglicht, einen Kunden teilweise oder vollständig vom Netz zu trennen. power liegt im Intervall zwischen 0 und 1 steht für den Anteil der maximalen Stromaufnahme, die das Smart-Meter dem Netz entnehmen darf, wobei 0 eine vollständige Netztrennung darstellt und 1 den normalen Modus repräsentiert. ts repräsentiert den Zeitpunkt, zu dem die Stromverbrauchsvorgabe aktiv wird. Die Vorgehensweise ist identisch zum vorhergehenden Protokoll:

1. GO → M: hi, init switch_power
2. M → GO: nonce n
3. GO → M: $\{[\text{switch\_power, M, n, ts, power}]_{GO}\}_M$

Wiederum entschlüsselt das Smart-Meter die Nachricht und überprüft die Signatur und die Zufallszahl n. Bei erfolgreicher Verifikation wird die Änderung umgesetzt.

Der letzte Nachrichtentyp stellt die Übertragung der Verbrauchsdaten an den Versorger (S) dar (meter_report). Meter readings entspricht dem aktuellen Zählerstand des Smart Meters, time der aktuellen Uhrzeit.

1. M → S: $\{[\text{M,time, meter readings}]_M\}_S$

Dieses grundlegende Protokoll stellt die Integrität und Verbindlichkeit der übertragenen Daten sicher, da durch Verwendung von digitalen Signaturen mögliche Veränderungen an den Daten erkannt werden und die Nachrichten an jedes Smart-Meter gebunden sind. Ebenfalls ist die Vertraulichkeit sichergestellt, da alle relevanten Informationen verschlüsselt werden und somit nur vom Empfänger entschlüsselt werden können. Ein Schutz vor Replay-Attacken ist zusätzlich gegeben, da durch die Zufallszahl (nonce) n jeweils unterschiedliche Nachrichten übermittelt werden. Der Datenschutz ist allerdings nicht sichergestellt, da der Versorger (bei zu gering gewählten Update-Intervallen) die in Kapitel 1.2 vorgestellten Rückschlüsse auf das Verhalten des Nutzers ziehen kann. Bei einheitlichen Tarifen und ausreichend groß gewählten Abrechnungszeiträumen (zum Beispiel monatlicher Rechnungsstellung) ist das Protokoll allerdings ausreichend. Falls jedoch tageszeitabhängige Tarife angeboten werden, so darf nicht der Verbrauch jeder einzelnen Tarifeinheit übermittelt werden, da dadurch wiederum Rückschlüsse auf das individuelle Verhalten des Nutzers möglich sind. So kann beispielsweise ermittelt werden, dass ein Nutzer zur Mittagszeit sehr wenig Strom benötigt und daraus geschlossen werden, dass er berufstätig ist und zu dieser Zeit das Haus verlassen ist. Das Protokoll sollte deshalb um weitere Nachrichten ergänzt werden, so dass der Energieversorger dem Smart Meter die aktuellen Preise pro Kilowattstunde (abhängig von der Tageszeit) mitteilen kann und das Smart Meter am Ende des Abrechnungszeitraums (zum Beispiel monatlich) die Gesamtkosten des Haushalts übermittelt.
Im folgenden Kapitel wird eine Erweiterung des Protokolls eingeführt, so dass auch die Abrechnung des Verbrauchs bei tageszeitabhängigen Stromtarifen möglich ist.

4.3 Grundlegendes Protokoll (erweitert)

Das grundlegende Protokoll wird um einen weiteren Protokollschritt (set_tariff) erweitert, in dem der aktuelle Energieversorger (S) dem Smart Meter (M) die aktuelle Tarifstruktur (tariff) und den Beginn der Gültigkeit (ts) mitteilt, damit nach Ablauf des Abrechnungsintervalls (P) die Kosten vom Smart Meter übermittelt werden können:

1. S → M: hi, init set_tariff
2. M → S: nonce n
3. S → M: $\{[\text{set\_tariff, M, n, S, tariff, ts, P}]_S\}_M$

Beim Empfang dieser Nachricht muss wiederum das Smart Meter überprüfen, ob die Signatur korrekt ist. Nur bei erfolgreicher Prüfung wird der neue Tarif angenommen und

doi: 10.2313/NET-2012-04-1_4

der Stromverbrauch des Haushalts entsprechend überwacht. Zusätzlich die wird die Übertragung des Verbrauchs (meter_report) angepasst, so dass nun nicht der Zählerstand übertragen wird, sondern die Kosten des angefallenen Stromverbrauchs (priced_meter_readings):

1. M → S: $\{[\text{M,time, priced\_meter\_readings}]_{\text{M}}\}_{\text{S}}$

Somit erfüllt das Protokoll alle Anforderungen für Abrechnungszwecke. Sämtliche Nachrichten sind digital signiert und verschlüsselt, somit ist sowohl die Integrität als auch die Authentizität sichergestellt und die Daten sind ohne Kenntnis des Schlüssels nicht zu entschlüsseln. Für die Lastermittlung ist ein solches Protokoll aber nicht geeignet, da alle Nachrichten des Smart Meters direkt dem Haushalt zuordenbar sind. Um eine Lastermittlung des Netzes bei gleichzeitigem Schutz der Privatsphäre zu ermöglichen, wird im folgenden Kapitel das No-Leakage Protokoll vorgestellt.

4.4 No-Leakage Protokoll

Das No-Leakage Protokoll ist nur zur Überwachung der Netzlast geeignet und geht davon aus, dass N (üblicherweise circa 100) Haushalte an einer Zwischenstation (SST) angeschlossen sind, die den Gesamtverbrauch dieser Haushalte erfasst (siehe Abbildung 2) und an den Netzbetreiber weiterleitet. Im Gegensatz zu einer ausschließlichen direkten Messung bei der Zwischenstation bietet es den Vorteil, dass dadurch erkannt werden kann, ob dem Stromnetz unbemerkt Strom entnommen wird. Falls der Netzbetreiber darauf verzichtet, wäre es auch denkbar, dass der Netzbetreiber spezielle Smart Meter bei jedem N-ten Kunden einbaut, die dann die Funktion der Zwischenstation übernehmen. Somit müsste der Netzbetreiber keine eigene Messstellen mehr betreiben. Damit könnte jedoch eine unbemerkte Stromentnahme nicht mehr entdeckt beziehungsweise lokalisiert werden.
Im No-Leakage Protokoll melden die Smart-Meter aller Haushalte ihren Verbrauch (m_i) gleichzeitig in regelmäßigen Abständen an diese Zwischenstation. Zur Verschlüsselung der Daten wird ein additiv-homomorphes Kryptosystem verwendet, das eine asymmetrische Verschlüsselung verwendet. Somit gilt Folgendes:

$$\{\text{m}_1\}_\text{k} * \{\text{m}_2\}_\text{k} = \{\text{m}_1 + \text{m}_2\}_\text{k}$$

Die Multiplikation zweier mit dem gleichen öffentlichen Schlüssel verschlüsselter Nachrichten entspricht also der Addition der beiden Klartexte (in dem Fall Stromverbräuche) und der anschließenden Verschlüsselung der Nachrichten. Das Paillier Kryptosystem besitzt beispielsweise diese Eigenschaften [10]. Zu Beginn des No-Leakage Protokolls verschickt die Zwischenstation die Zertifikate aller teilnehmenden Meter an alle Meter (1). Anschließend werden von jedem Meter N Zufallszahlen so gewählt, dass die Summe der Zufallszahlen (Modulo n, wobei n groß genug gewählt wird) dem eigenen Verbrauch in diesem Zeitraum entspricht. N-1 Zufallszahlen werden mit den N-1 öffentlichen Schlüsseln der restlichen beteiligten Smart-Metern verschlüsselt und an die Zwischenstation geschickt (2). Die Zwischenstation multipliziert die Nachrichten mit identischen Schlüsseln und versendet das Ergebnis an diejenige Station, die den privaten Schlüssel für die jeweilige Nachricht besitzt (3). Nun entschlüsselt jede Station die Nachricht und addiert die noch nicht versendete Zufallszahl auf die Nachricht und sendet das Ergebnis in Klartext an die Zwischenstation (4), die nun überprüfen kann, ob der übermittelte Gesamtverbrauch dem selbst gemessen Gesamtverbrauch entspricht. Der Ablauf des Protokolls ist im Folgenden dargestellt:

1. SST → M_i: no-leakage, cert_{M_1},...,$\text{cert}_{\text{M}_\text{N}}$
2. M_i → SST: $\text{y}_{\text{i}1}$,...,$\text{y}_{\text{ii}-1}$,$\text{y}_{\text{ii}+1}$,...,y_{iN}
   wobei M_i Zufallszahlen $\text{a}_{\text{i}1}$,...,a_{iN} wählt,
   so dass $\text{m}_\text{i} = \sum_{\text{j}=1}^{\text{N}} \text{a}_{\text{ij}} \bmod n$
   mit $\text{y}_{\text{ij}} := \{\text{a}_{\text{ij}}\}_{\text{pk}_\text{j}}$ und $\text{j} \in \{\ \text{j} \in [\text{N}] \mid \text{j} \neq \text{i}\ \}$
3. SST → M_i: $\prod_{j \in \{j \in [N] | j \neq i\}} y_{ji} = \left\{ \sum_{j \in \{j \in [N] | j \neq i\}} a_{ji} \right\}_{pk_i}$
4. M_i → SST:
   $$\sum_{j \in \{j \in [N] | j \neq i\}} a_{ji} + a_{ii} = \sum_{j} a_{ji} \bmod n$$

Insgesamt werden dabei 4N Nachrichten verschickt (in jedem Schritt N Nachrichten). Die Zwischenstation kann nicht auf den Verbrauch einzelner Haushalte schließen, da sie nur Summen des Verbrauchs aller beteiligten Haushalte unverschlüsselt übermittelt bekommt. Es kann sogar gezeigt werden, dass ein Angreifer nicht einmal die Vertauschung des Stromverbrauchs zweier beliebiger Haushalte erkennen kann. Somit kann mit Sicherheit auch nicht auf den Verbrauch einzelner Haushalte geschlossen werden. Das Protokoll ist dann sicher, wenn sich zumindest zwei Haushalte nach der Protokollspezifikation verhalten. Wenn ein Angreifer beispielsweise den Gesamtverbrauch und den Verbrauch von allen bis auf einen Haushalt kennt, kann er trivialerweise den Verbrauch des verbleibenden Haushalts ermitteln, in dem er die Verbräuche der bekannten Haushalte vom Gesamtverbrauch subtrahiert. Da N jedoch in der Größenordnung von 100 liegen sollte, stellt dies keine größere Gefahr dar. Ebenfalls geht das Protokoll davon aus, dass kein Angreifer eine große Anzahl an privaten Schlüsseln der verwendeten Zertifikate ermitteln kann. Diese Annahme wird jedoch bei allen Protokollen getroffen, die auf asymmetrischer Verschlüsselung in Form von Zertifikaten basieren und stellt somit keine Einschränkung dar, so lange in der Implementierung darauf geachtet wird, dass ausreichend lange Schlüssel (>2048 Bit) verwendet werden und/oder die Schlüssel regelmäßig erneuert werden.

5. ZUSAMMENFASSUNG

Insgesamt existieren, wie in Kapitel 4 gezeigt, Möglichkeiten, um den Privatsphäre von Nutzern zu schützen. Die Energieversorger können damit sowohl flexible Tarifstrukturen anbieten und haben bei gleichzeitigem Schutz der Privatsphäre der Nutzer die Möglichkeit, die Netzlast zu messen. In aktuellen Geräten sind diese Funktionen bislang noch nicht implementiert. So übertragen die in [11] verwendeten Messgeräte die Verbrauchswerte unverschlüsselt und mit Geräte-ID und ermöglichen so einerseits eine Aggregation der Daten für einzelne Benutzer, als auch möglicherweise eine Identifikation der Nutzer und sind damit absolut nicht geeignet, um die Privatsphäre der Verbraucher zu schützen. Auch die

doi: 10.2313/NET-2012-04-1_4

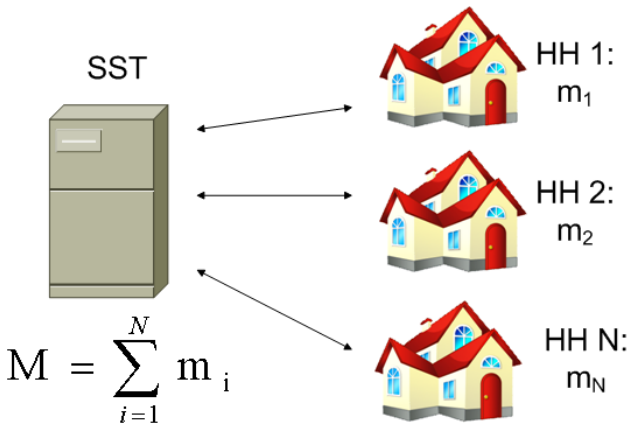


Abbildung 2: Haushalte (HH) melden Stromverbrauch (m_i) an Zwischenstation (SST)

aktuell von den Energieversorgern angebotenen Tarife beinhalten das Risiko, dass detaillierte Verbrauchsdaten durch Angreifer ermittelt werden können, da die Verbrauchsdaten ins Internet übertragen werden, um den Nutzern einen mobilen Zugriff zu ermöglichen. Für besseren Datenschutz wäre es jedoch wünschenswert, wenn eine Abfrage nur innerhalb des eigenen lokalen Netzwerks möglich ist, und somit nicht das Risiko besteht, dass Verbrauchsdaten in fremde Hände geraten. Einen mobilen Abruf könnte man dennoch beispielsweise per VPN realisieren.
Somit spricht aus Sicht des Datenschutzes bei Verwendung sinnvoller Kommunikationsprotokolle zwischen Smart Meter und Netzbetreiber beziehungsweise Stromversorger nichts gegen die Einführung von intelligenten Stromzählern, die den Stromverbrauch (für Abrechnungszwecke und zur Lastermittlung) selbstständig übertragen. Besonders aufgrund der Möglichkeiten, den Zustand des Stromnetzes zu überwachen, ist zu erwarten, dass diese Geräte in naher Zukunft verpflichtend eingeführt werden. Auch bei der ab 2015 vorgeschriebenen monatlichen Abrechnung des Stromverbrauchs auf Grundlage des tatsächlichen Verbrauchs bieten sie für Kunden einen deutlich größeren Komfort, da die Verbrauchsdaten nicht manuell an den Stromversorger übermittelt werden müssen.

6. LITERATUR

[1] Privatkunden - intelligenter stromzähler - faq. Website. http://www.enbw.com/content/de/privatkunden/produkte/zusatzinformationen/isz_faq/index.jsp.

[2] R. Anderson and S. Fuloria. On the Security Economics of Electricity Metering. In *Proceedings of the Ninth Workshop on the Economics of Information Security (WEIS)*, June 2010.

[3] N. T. A. A. Dr. Helge Kreutzmann, Stefan Vollmer. Protection profile for the gateway of a smart metering system. Technical report, Bundesamt für Sicherheit in der Informationstechnik (BSI), 2011.

[4] E. Energie. Ex-post daten archiv 2010. http://www.eon-schafft-transparenz.de/download/expost, 2011.

[5] EON. E.on energienavi. Website. https://www.eon.de/de/eonde/pk/produkteUndPreise/Strom/E.ON_Energienavi/index.htm.

[6] O. (France). *Photovoltaic Barometer*. EurObserv'ER, 2011.

[7] F. D. Garcia and B. Jacobs. Privacy-friendly energy-metering via homomorphic encryption. In J. C. et al., editor, *6th Workshop on Security and Trust Management (STM 2010)*, volume 6710 of *Lecture Notes in Computer Science*, pages 226–238. Springer Verlag, 2010.

[8] J. Hladjk. Smart metering und eu-datenschutzrecht. *Datenschutz und Datensicherheit - DuD*, 35:552–557, 2011. 10.1007/s11623-011-0136-5.

[9] T. Jeske. Datenschutzfreundliches smart metering. *Datenschutz und Datensicherheit - DuD*, 35:530–534, 2011. 10.1007/s11623-011-0132-9.

[10] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT*, pages 223–238, 1999.

[11] D. L. M. Prof. Dr.-Ing U. Greveler, Dr. B. Justus. Hintergrund und experimentelle ergebnisse zum thema smart meter und datenschutz. Technical report, Labor für IT-Sicherheit der FH Münster, 2011.

[12] RWE. Smart meter - inteligente mess- und zähltechnik von morgen. Website. http://www.rwe.com/web/cms/de/238130/rwe/innovationen/energieanwendung/smart-meter/.

doi: 10.2313/NET-2012-04-1_4

Einführung in Netzwerk-Fluss-Probleme Das min-cut max-flow Theorem

Felix Meyner
Betreuer: Stephan Günther
Hauptseminar Innovative Internet-Technologien und Mobilkommunikation WS 11/12
Lehrstuhl Netzarchitekturen und Netzdienste
Fakultät für Informatik, Technische Universität München
E-Mail: felix.meyner@mytum.de

KURZFASSUNG

Graphen lassen sich für eine Vielzahl von Problembeschreibungen und deren Lösung einsetzen. In diesem Paper wird der Schwerpunkt auf Flüssen in Netzwerken liegen. Eines der in diesem Kontext auftretenden Probleme ist die Bestimmung des maximal möglichen Flusses in einem Netzwerk. Diese Herleitung wird im Folgenden Schritt für Schritt dargelegt. Ziel dieser Arbeit ist es zu zeigen, dass eine starke Dualität zwischen dem Schnitt des Netzwerkes mit der kleinsten Kapazität und dem maximal möglichen Fluss existiert. Somit können beide Optimierungsprobleme herangezogen werden, um den maximalen Fluss zu berechnen.

Schlüsselworte

Netzwerk-Fluss-Problem, Graphentheorie, Exzess, Fluss, Augmentation, Sättigung, Kapazität, Ford-Fulkerson Algorithmus, min-cut-max-flow, Transformation

1. EINLEITUNG

Problembeschreibungen in Graphen lassen sich generell in zwei Kategorien unterteilen: Maximierungsprobleme und Minimierungsprobleme. Ziel dieses Papers ist es, die Dualität dieser beiden Problemarten am Beispiel des maximal möglichen Flusses aufzuzeigen. Somit soll gezeigt werden, dass der minimale Schnitt eines Netzwerkes dem maximale Fluss entspricht.
Um dies zu beweisen, ist es notwendig zuerst die Grundbegriffe und deren mathematische Herleitung zu erklären, um anschließend diese Dualität mit einem Ringbeweis zu belegen. Darüber hinaus wird ein Überlick über mögliche Transformationen von Problemen zur Vereinfachung gegeben.

2. DEFINITION NETZWERK

Hinführend hierfür werden im Folgenden die wichtigsten Begriffe eingeführt und kurz erklärt, damit ein gemeinsames Verständnis der verwendeten Begriffe geschaffen werden kann.

2.1 Der Graph

Um den Beweis der Dualität führen zu können wird ein Netzwerk als $N = (G, c, q, s)$ definiert. Hierbei ist G ein endlich gerichteter Graph $G = (V, E)$ mit den Knoten $v \in V(G)$, wobei die Knoten $q \in V$ die Quelle und $s \in V$ die Senke des Netzwerks darstellen. Jede gerichtete Kante $e \in E(G)$ verbindet zwei Knoten miteinander. Jede gerichtete Kante $e = (v, w)$, wobei $v, w \in V(G)$, besitzt eine maximale Kapazität $c(v, w) > 0$. Da die Kapazität jeder Kante $c(e) > 0$ ist folgt:

Definition 2.1:
Die Gesamtkapazität eines Graphen ist > 0.

$$\sum_{e \in E(G)} \mathrm{c(e)}{>}0 \qquad \Rightarrow \qquad c(G) \rightarrow \mathbb{R}^+$$

2.2 Fluss

Über jede Kante $e(v, w)$ im Graphen kann ein Fluss $f(v, w)$ fließen. Dieser Fluss muss aufgrund der gerichteten Kante positiver Natur sein. Es ist jedoch möglich, dass eine Kante keinen Fluss besitzt. Somit gilt:

Definition 2.2:
Der Gesamtfluss in einem Graphen ist ≥ 0.

$$\sum_{e \in E(G)} f(e) \geq 0 \qquad \Rightarrow \qquad f(G) \rightarrow \mathbb{R}_0^+$$

Nun führen wir die unter ***2.1*** definierte Kapazität mit der gerade definierten Flussmenge zusammen und definieren Folgendes:

Definition 2.3: Kapazitätskonformität
Der Fluss über eine Kante kann nicht größer sein, als die Kapazität der Kante.

$$\forall (v, w) \in E(G) : f(v, w) \leq c(v, w) \qquad \text{(I)}$$

Der Vollständigkeit halber wird ebenfalls Folgendes für jede nicht existierende Kante definiert:

$$(v, w) \notin E(G) : f(v, w), c(v, w) = 0$$

Der Fluss einer Kante $f(e)$ wird über eine Rückkante $\overleftarrow{e}$ abgebildet. Für die Rückkante gilt:

$$\overleftarrow{e}(w, v) = f(v, w) : f(e) > 0$$

Da die Knoten des Graphen über Kanten verbunden sind, soll nun definiert werden wie sich Knoten in einem Netzwerk verhalten. Hierzu sind folgende Grundannahmen zu treffen, die für die Knoten $v \in V(G) \setminus \{q, s\}$ des Graphen gelten:

doi: 10.2313/NET-2012-04-1_5

1. Knoten verbrauchen keine Ressourcen.
2. Knoten vermehren keine Ressourcen.
3. Knoten haben keine Möglichkeit Ressourcen zu speichern.

Daraus kann man ableiten, dass der Fluss in einen Knoten dem Fluss aus einem Knoten entsprechen muss. Formal definieren wir nun folgendes:

δ^+als den Fluss in einen Knoten:

$$\delta^+(v) \coloneqq \{e = (w, v) \in E \mid w \in V\}$$

δ^- als den Fluss aus einem Knoten:

$$\delta^-(v) \coloneqq \{e = (v, w) \in E \mid w \in V\}$$

Definition 2.4: Flusserhaltungsbedingung
Der Fluss in einen Knoten entspricht dem Fluss aus einem Knoten, wobei die Quelle und die Senke ausgenommen sind.

$$\forall v \in V(G) \setminus \{q, s\} : \sum_{(w,v)\in\delta^+(v)} f(w, v) = \sum_{(v,w)\in\delta^-(v)} f(v, w) \quad \text{(II)}$$

Die Quelle und Senke nehmen in einem endlichen Graphen eine Sonderrolle ein, da die Flusserhaltung für diese nicht gilt. Der Grund hierfür ist, dass es keine Zirkulation zwischen der Senke und der Quelle gibt. Betrachtet man nun die Senke eines Graphen, so gilt für diese folgendes:

$\delta^+(s) = \sum_{(v,s)\in E(G)} f(v, s)$
$\delta^-(s) = 0$

Es besteht somit ein Überschuss in der Senke. Dieser Überschuss wird als Exzess definiert:

Definition 2.5:
Der Exzess eines Knotens entspricht dem Überschuss des Flusses in diesem.

$$v_{exz} = \delta^+(v) - \delta^-(v)$$

2.3 Residualkapazität

Wenn die Kapazität einer Kante noch nicht gesättigt ist, dies bedeutet dass $c(e) - f(e) > 0$ ist, so bezeichnet man diese Kante als augmentierend. Sie besitzt folglich noch weitere Restkapazitäten die aufgenommen werden können [10].

Allgemein kann man diese Restkapazitäten oder Residualkapazitäten über eine Rückkante $\overleftarrow{e} = (w, v) : f(v, w) > 0$ als $\overleftarrow{e}(w, v) = f(v, w)$ darstellen.

Definition 2.6:
Die Residualkapazität einer Kante entspricht der möglichen Flusserweiterung über diese unter Beachtung von (I) *und* (II)

$$c_f(v, w) = c(v, w) - f(v, w) + f(w, v)$$

2.4 Pfad

Ein Pfad $P(G)$ von der Quelle q zur Senke s beschreibt eine Menge von Kanten, die aneinandergereiht eine der möglichen Verbindungen $p(q, s)$ darstellen. Somit gilt: $p(q, s) \in P(q, s)$. Für Pfade gelten die selben Bedingungen in Bezug auf die Kapazität und den Fluss, wie für die singulär enthaltenen Kanten dieses Pfades. Somit darf der Fluss über einen Pfad nicht größer sein, als die kleinste Kapazität der beinhalteten Kanten. Also gilt:

$$max\, f(p) \leq min\, c(v, w) \in p(q, s)$$

2.5 Schnitt

Als letzter grundlegender Bestandteil wird nun der Schnitt eines Netzwerks definiert. Ein (q,s)-Schnitt $a(G)$ eines Graphen $G = (V, E)$ teilt diesen in zwei Partitionen $(Q, Q - S)$. Es wird festgelegt, dass $q \in Q$ und $s \in S$ sei. Ein Graph besitzt hierdurch eine endliche Anzahl an möglichen Schnitten $A(G)$.

Definition 2.7:
Ein Schnitt $a(G)$ teilt einen Graphen in zwei Partitionen. Diese sind definiert durch (1.) die Knoten der Partitionen und (2.) durch die durchtrennten Kanten des Schnitts [10]:

1. $V_a(Q, S) : Q \cup S = V(G) \wedge Q \cap S = \emptyset$
2. $E_a = \{(v, w) \in E \mid v \in Q, w \in S\}$

Die Kapazität eines Schnitts $a(G)$ bestimmt sich folglich aus der Summe der Kapazitäten der durchtrennten Kanten.

$$c_a = \sum_{v\in S \wedge w\in Q} c(v, w)$$

Analog kann man den Fluss eines Schnitts wie folgend definieren:

$$f_a = \sum_{v\in S \wedge w\in Q} f(v, w)$$

Da die Kapazitätskonformität (*Definition 2.3*) auch hier ihre Gültigkeit behält, darf auch der Fluss eines Schnittes nicht größer sein als seine Kapazität.

$$\forall a \in A : f_a \leq c_a$$

Final kann nun gesagt werden, dass beide Schnitte mit der minimalen Knotenmenge $Q \subset V(G) \setminus \{s\}$ oder $S \subset V(G) \setminus \{q\}$ auf jeden Fall den Fluss im Netzwerk besitzen.

Somit gelten folgende Annahmen:

$$\begin{aligned} f(G) &= \sum_{\exists(q,v)\in E(G)} f(q, v) = \mid q_{exz} \mid \\ &= f(Q) \\ &= \sum_{\exists(v,s)\in E(G)} f(v, s) = s_{exz} \\ &= f(S) \\ &\leq c_a \end{aligned}$$

doi: 10.2313/NET-2012-04-1_5

3. MAXIMALER FLUSS

Durch die bisher gezeigten Grundannahmen lässt sich bereits Folgendes zur Bestimmung des maximalen Flusses, in Form eines schwach dualen Problems, heranziehen:

$$max\, f(G) \leq min\, c_a(G)$$

Ziel ist es jedoch den maximal Fluss nicht als Näherungswert zu bestimmen, sondern diesen exakt zu berechnen. Hierfür kann ein einfacher Ringbeweis genutzt werden. Wenn man folgende Annahmen hierzu heranzieht [2]:

Annahmen 3.1:
Folgende Aussagen sind äquivalent.

1. Es gibt im Netzwerk N einen maximalen Fluss f.
2. Es gibt im Residualnetzwerk N_f keinen augmentierenden Pfad.
3. Der Fluss des minimalen $f_a(Q,S)$ entspricht dem maximalen Fluss $f(G) = f_a(Q,S)$

Beweis 3.2:
(1) ⇒ (2) :
Wenn es in dem Residualgraphen G_f einen augmentierenden Pfad gibt, so ist der Fluss in diesem Residualgraphen nicht maximal, da $f = f + f(G_f)$. Dies widerspricht der *1. Annahme*, dass der Fluss maximal ist.
(2) ⇒ (3) :
Wenn es in einem Residualgraphen keinen augmentierenden Pfad mehr gibt, so existiert ein natürlicher Schnitt in diesem Graphen, der die Quelle und die Senke voneinander trennt.Gekennzeichnet werden die Partitionen durch die gesättigten Kanten.Vergleiche ***Definition 2.7 Punkt 2***. Somit können folgende Annahmen getätigt werden: Der Fluss in diesem natürlichen Schnitt muss der Kapazität dieses Schnitts entsprechen. Formal: $f(G) = f_a(Q,S) = c_a(Q,S)$.
(3) ⇒ (1):
Da der Fluss in jedem Schnitt f_a durch die jeweilige Kapazität c_a nach oben begrenzt ist, muss in mindestens einem Schnitt die Kapazität gleich dem maximalen Fluss sein, da es sonst kein maximaler Fluss wäre, da eine Erhöhung des Flusses möglich ist.

Dieser geführte Beweis ist der Beleg des min-cut max-flow Theorems. Dieses besagt, dass der maximal mögliche Fluss durch einen Graphen dem minimalen Schnitt entspricht.

Definition 2.8 Min-cut max-Flow:
Der maximale Fluss in einem Graphen entspricht der minimalen Kapazität der möglichen Schnitte des Graphen

$$min\, c_a(G) = max\, f(G)$$

Hierzu gilt jedoch anzumerken, dass es auch mehrere minimale Schnitte in einem Graphen geben kann und somit der maximale Fluss einem minimalen Schnitt entspricht.

4. FORD-FULKERSON ALGORITHMUS

Diese drei Annahmen macht sich der Ford-Fulkerson-Algorithmus zu nutze. Entwickelt wurde dieser 1956 von L.R. Ford und D.R. Fulkerson [3]. Der Algorithmus terminiert nach endlich vielen Schritten unter der Bedingung, dass die Kapazitäten der Kanten keine irrationalen Zahlen sind. Falls dies der Fall ist, kann es sein, dass der Algorithmus nicht terminiert. Folgende Annahmen sind aus den Artikeln [4][3] und dem Buch [5] entnommen.

4.1 Formale Beschreibung

Der beschriebene Algorithmus ist ein iteratives Verfahren, um den maximalen Fluss in einem Netzwerk zu bestimmen. Hierfür wird folgender Ablauf verwendet:

Initialisierung: Setze alle Flüsse der Kanten auf ein gültiges Minimum zurück.

$$\Rightarrow f(e) = 0 \text{ für alle } e \in E(G)$$

Solange es augmentierende Pfade von q nach s gibt:

1. Wähle einen beliebigen Pfad $p(q,s)$ wobei: $e(p) \in E(G)$
2. Bestimme die minimale Residualkapazität κ aller $e(p)$
$\kappa := min\{c_f(e) \mid e \in p\}$
3.1 Für alle $e \in p : f(e) = f(e) + \kappa$
3.2 Für alle $\overleftarrow{e} \in p : f(e) = f(e) - \kappa$

Eine wichtige Eigenschaft dieses Algotithmus ist es, dass das Ergebnis des maximalen Fluss deterministisch ist. Jedoch kann der gefundene minimale Schnitt unter der Vorraussetzung mehrere Schnitte mit der selben minimale Kapazität variieren. In diesem Fall hängt der gefundene minimale Schnitt von den gewählten Pfaden ab. Somit eignet sich der originäre Algorithmus nur dazu, den maximalen Fluss eines Netzwerks zu bestimmen.

4.2 Beispiel

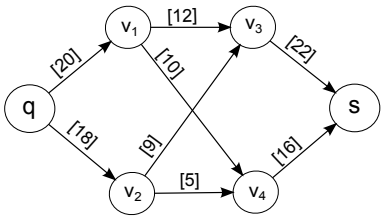


Abbildung 1: Basisgraph

Nun wird der Ford-Fulkerson Algorithmus an einem Beispiel *(Abbildung 1)* gezeigt.
Der Formalität halber wird vor der Quelle q eine virtuelle Superquelle + eingeführt. Diese dient dazu in der Tableaudarstellung den Knoten q korrekt abbilden zu können. Ein weiterer Grund Superquellen einzuführen wird in **Kapitel 5.1.3** aufgeführt.
Es gilt für +:

$$c(+,q) = \infty$$

doi: 10.2313/NET-2012-04-1_5

In der Tableaudarstellung *(Tabelle 1)* wird folgende Notation verwendet:
Es sind unter $V(G)$ alle realen Knoten des Graphen abgebildet.
Für jede Iteration wird eine neue Spalte erzeugt. Diese bildet den gewählten Pfad ab.
Wenn ein Knoten auf dem gewählten Pfad liegt, wird bei diesem der Vorgängerknoten mit der Residualkapazität der verbindenden Kante eingetragen.

(Vorgängerknoten, Residualkapazität der Kante)

Anschließend wird die kleinste Residualkapazität gewählt und die Residualkapazitäten aller Kanten $\in p$ um diesen Fluss verringert. Zuletzt werden die entsprechenden Rückkanten in den Graphen eingetragen. Zu sehen ist dies exemplarisch an *Abbildung 2*, welche den Zustand des Graphen nach der ersten Iteration zeigt.

Tabelle 1: Tableau Ford-Fulkerson-Algorithmus

$V(G)$	Initialisierung	1.Schritt	2.Schritt	3.Schritt	4.Schritt
q	$(+,\infty)$	$(+,\infty)$	$(+,\infty)$	$(+,\infty)$	-
v_1	$(q^+,0)$	$(q^+,20)$	-	$(q^+,8)$	-
v_2	$(q^+,0)$	-	$(q^+,18)$	-	$(q^+,13)$
v_3	$(v_1^+,0)(v_2^+,0)$	$(v_1^+,12)$	-	-	$(v_2^+,9)$
v_4	$(v_2^+,0)(v_1^+,0)$	-	$(v_2^+,5)$	$(v_1^+,10)$	-
s	$(v_3^+,0)(v_4^+,0)$	(v_3^+22)	$(v_4^+,16)$	$(v_4^+,11)$	(v_3^+10)
Δ	-	12	5	8	9
Σ	0	12	17	25	34

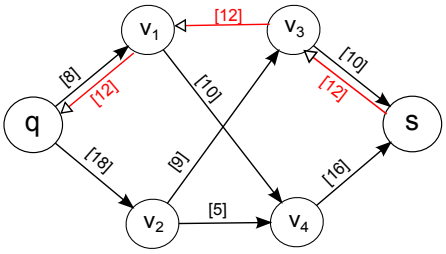


Abbildung 2: Zustand nach 1. Iteration

Initialisierung

Alle Flüsse werden mit einem gültigen Startfluss initialisiert. In diesem Fall ist 0 der gewählte Startfluss.

1. Iteration

Es wird ein beliebiger Pfad p(Q,S) gewählt.
$p_1(q,s) = (q,v_1),(v_1,v_3),(v_3,s)$
Die kleinste Residualkapazizät in p_1 ist auf der Kante $e(v_1,v_3)$ mit 12 Einheiten. Wie auf *Abbildung 2* zu sehen ist, wurden alle Residualkapazitäten um 12 Einheiten verringert und die entsprechenden Rückkanten eingetragen. Die Kante $e(v_1,v_3)$ ist somit saturiert und fällt weg.

2. Iteration

$p_2(q,s) = (q,v_2),(v_2,v_4),(v_4,s)$
$min\, c(p_2) = c(v_2,v_4) = 5$

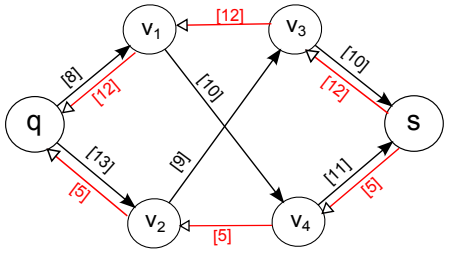


Abbildung 3: Zustand nach 2. Iteration

3. Iteration

$p_3(q,s) = (q,v_1),(v_1,v_4),(v_4,s)$
$min\, c(p_3) = c(q,v_1) = 8$

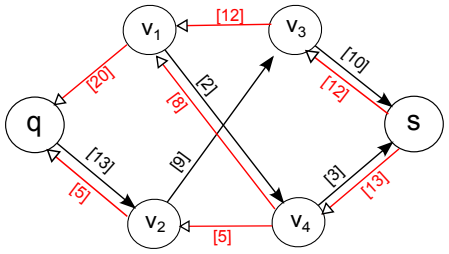


Abbildung 4: Zustand nach 3. Iteration

4. Iteration

$p_4(q,s) = (q,v_2),(v_2,v_3),(v_3,s)$
$min\, c(p_3) = c(v_2,v_3) = 9$

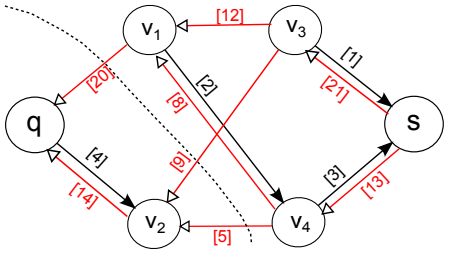


Abbildung 5: Zustand nach 4. Iteration mit gefundenem minimalen Schnitt

Bei Betrachtung der *Abbildung 5* ist erkennbar, dass es keinen weiteren augmentierenden Pfad von der Quelle zur Senke mehr gibt. Der Algorithmus terminiert nun und hat den maximalen Fluss von 34 Einheiten erreicht. Durch die eingetragenen Rückkanten ist ebenfalls der minimale Schnitt klar erkennbar.

In diesem Fall umfasst der minimale Schnitt:

$Q = \{q, v_2\}\,,\ S = \{v_1.v_3, v_4, s\}$
$E_a = \{(v,w) \in E \mid v \in Q, w \in S\}$

$$f(G) = f(E_a) = \sum_{e \in E_a} f(e) = \sum_{e \in E_a} c(e) = min\, c_a(Q,S) = 34$$

doi: 10.2313/NET-2012-04-1_5

5. GRENZEN UND LÖSUNGSANSATZ

Die Herleitungen und Definitionen wurden bis jetzt bewusst sehr allgemein gehalten, damit ein breiteres Anwendungsgebiet abgedeckt werden kann. Jedoch liegen in der Realität Problemstellungen meist nicht in den vorrausgesetzten Formaten vor.

5.1 Transformationen

Um real auftretende Problemstellungen nun auf diese Verallgemeinerungen zu überführen kann man folgende Transformationen verwenden.

5.1.1. Ungerichtete Kanten

Unter "***2.1 Der Graph***" wurde ein Graph als gerichtet definiert. Dies bedeutet, dass jede Kante eine Flussrichtung besitzen muss. Falls ein Graph jedoch ungerichtet ist, können die Definitionen nicht ohne weiteres übernommen werden. Um die mathematischen Beweise in der einfacheren Form weiternutzen zu können, kann man eine ungerichtete Kante in eine gerichtete Kante überführen. Dies ist auf *Abbildung 6* zu sehen. Die Residualkapazitäten der gerichteten Kanten $c_{f\,gerichtet}(v_1, v_2)$,c($v_2$,$v_1$) verhalten sich zur Kapazität der ungerichteten Kante $c_{f\,ungerichtet}\{v_1, v_2\}$:

$$c_{f\,ungerichtet}(e) = c(e_{ungerichtet}) - \sum_{e \in E_{gerichtet}} f(e)$$

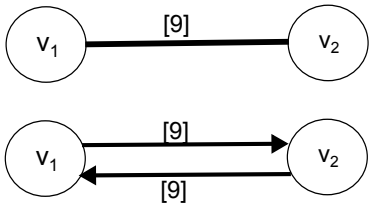


Abbildung 6: Transformation ungerichtete Kante

5.1.2. Kapazitätsbegrenzte Knoten

Falls ein Knoten eine Kapazitätsbegrenzung aufweist, kann dies gelöst werden indem der Knoten v in zwei Knoten v_{in} und v_{out} überführt wird und für die Kapazität gilt, dass [8]:

$$c(v) = c(v_{in}, v_{out})$$

Dies wird in *Abbildung 7* veranschaulicht.

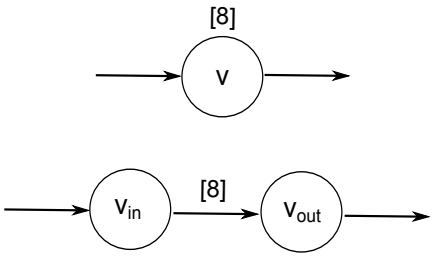


Abbildung 7: Transformation kapazitätsbeschränkter Knoten

5.1.3. Mehrere Quellen und Senken

Ebenso kommt es vor, dass es in einem Netzwerk mehrere Quellen und/oder Senken gibt (*Abbildung 8*).

Diese Problematik kann bei einem homogenen Fluss gelöst werden, indem man eine Superquelle " + ", welche bei den Quellen als Vorgängerknoten ergänzt wird, oder eine Supersenke "−" hinzufügt, welche jeweils die Senken als nachfolgender Knoten erweitert. Die Kapazitäten der neuen Kanten werden auf ∞ gesetzt (*Abbildung 9*).

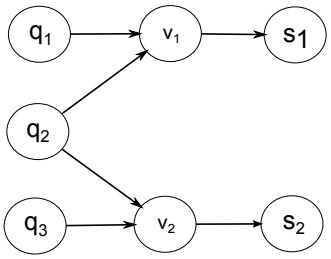


Abbildung 8: Netzwerk mit multiplen Quellen und Senken

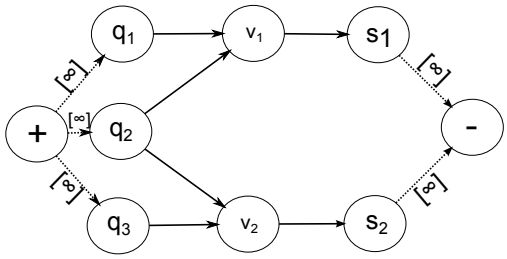


Abbildung 9: Superquellen und Supersenken

5.2 Anwendbarkeit

Die bis hier vorgestellten Methoden lassen sich bisher nur bedingt in der Netzwerktechnik einsetzen. Die meisten Routingprotokolle unterstützen momentan nur solche Methoden, die entweder den kürzesten oder den schnellsten Pfad suchen. Ein Routing über verschiedene Pfade führt, vorallem bei unterschiedlichen Delays der verschiedenen Pfade, zu Problemen, da die Reihenfolge der ankommenden Pakete von entscheidender Wichtigkeit ist.

Eine weitere Grenze der bisherigen Anwendung ist folgender Fall (*Abbildung 10*), in dem es konkurrierende Ströme gibt:
In einem Netzwerk gibt es zwei Quellen q_A und q_B wobei der Index bestimmt, welches Gut von der jeweiligen Quelle verschickt wird. Dazu gibt es zwei Senken die beide Güter benötigen.
Wie zu sehen ist, ist dieses Problem mit den bisherigen Mitteln nicht lösbar, da die Kante (v_1, v_2) die Engstelle darstellt. Unter der Annahme der Flusserhaltung kann der Knoten v_1 nur A oder B erhalten und somit wird eine der Senken nicht mit beiden Gütern versorgt.

Den Lösungsansatz zu diesem Problem bezeichnet man als Netzwerkcodierung [6].
Hierbei werden zuerst die Flüsse aus den Quellen als Informationen interpretiert. Dazu erhalten Knoten die Fähigkeit Informationen zu

doi: 10.2313/NET-2012-04-1_5

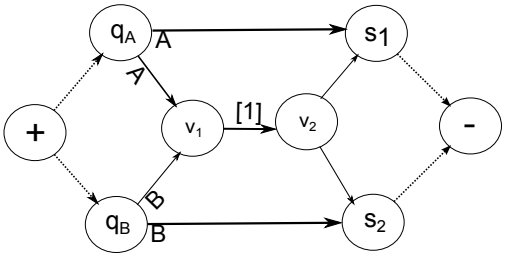


Abbildung 10: Konkurrierende Ströme

codieren und decodieren. In diesem Beispiel ist es der Knoten v_1, der beide Informationen mittels eines XOR Befehls zu $A \oplus B$ codieren kann.
Die Flusserhaltung ist in diesem Fall gegeben, da A und B in den Knoten fließen und $A \oplus B$ den Knoten verlässt. Somit sind beide Informationsflüsse gewährleistet. Die Senken können nun die ihnen fehlende Information aus dem $A \oplus B$ mittels der ihnen vorliegenden nicht codierten Information wieder extrahieren. Das gelöste Problem ist in *Abbildung 11* zu sehen.

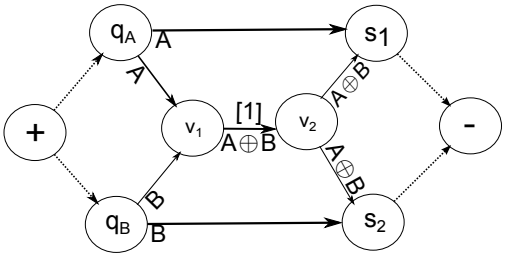


Abbildung 11: Konkurrierende Ströme

6. ZUSAMMENFASSUNG

Zusammenfassend kann man sagen, dass es theoretisch möglich ist den maximalen Fluss in einem Netzwerk zu bestimmen. In der Praxis wird dies auch zum Beispiel in der Logistik und bei homogenen Gütern wie Strom oder Wasser genutzt, um eine bestehende Infrastruktur so effizient wie möglich zu nutzen oder mit minimalem Aufwand zu erweitern.

In der Informationstechnologie liegen meist andere Anforderungen vor, die erfüllt werden müssen. Dies führt dazu, dass die aktuellen Standards mit den jetzigen technischen Anforderungen nicht versuchen den maximal möglichen Fluss zu finden. Der Grund hierfür ist, dass die Verwendung mehrere Pfade relativ problematisch ist, da es zu Auflösungsproblemen kommen kann. Ebenso erfordern die meisten Standards das Pakete in geordneter Reihenfolge ankommen. Würde man nun verschiedene Pfade mit unterschiedlichen Delays nutzen, wäre ein Rechenmehraufwand notwendig um die Pakete zu puffern, sie zu ordnen und vor allem wäre die Fehlerkorrektur erheblich aufwendiger. Da aber die Rechenkapazitäten sehr schnell wachsen ist es abzuwarten, wie sich die verwendeten Protokolle in Zukunft entwickeln werden.

7. LITERATUR

[1] T. H. Cormen, C. E. Leierson, R. Rivest, and C. Stein. *Algorithmen - Eine Einführung*. Oldenbourg, 2010.

[2] S. Dietzel. *Ford-Fulkerson-Methode zur Berechnung des maximalen Flusses in Graphen*. Juni 2005.

[3] L. R. Ford and D. R. Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics*, 8:399–404, 1956.

[4] L. R. Ford and D. R. Fulkerson. A simple algorithm for finding maximal network flows and an application to the hitchcock problem. *Canadian Journal of Mathematics*, 9:210–218, 1957.

[5] L. R. Ford and D. R. Fulkerson. *Flows in Networks, With a new foreword by Robert G. Bland and James B. Orlin*. Princeton Press, 2010.

[6] S. M. Günther. Optimal cost multicast over coded networks. Ferienakademie Sarntal, 2009.

[7] N. J. Harvey. Comparing network coding with multicommodity flow for the k-pairs communication problem. MIT-LCS-TR-964, 11 2004.

[8] T. H. Thalwitzer. Max-flow min-cut. Master's thesis, Universität Wien, 2008.

[9] L. Trevisan. Lecture 15. the linear programming formulation of maximum cut and its dual. Stanford University CS261: Optimization and Algorithmic Paradigms coursewebsite, 02 2011.

[10] L. Trevisan. Lecture 9. the maximum flow - minimum cut theorem. Stanford University CS261: Optimization and Algorithmic Paradigms coursewebsite, 02 2011.

doi: 10.2313/NET-2012-04-1_5

TETRA und seine Anwendung bei Behörden und Organisationen mit Sicherheitsaufgaben

Daniel Baumann
Betreuer: Stephan Günther
Hauptseminar - Innovative Internettechnologien und Mobilkommunikation WS2011
Lehrstuhl für Netzarchitekturen und Netzdienste
Fakultät für Informatik, Technische Universität München
Email: daniel.baumann@in.tum.de

KURZFASSUNG

Dieses Paper behandelt den Bündelfunk-Standard TETRA und seine Anwendung bei Behörden und Organisationen mit Sicherheitsaufgaben. Dabei wird TETRA beschrieben und auf Layer 3 Nachrichten eingegangen, welche bei einem Gespräch entstehen. Des Weiteren werden Gründe diskutiert, weswegen der Analogfunk nicht mehr dem Stand der Technik entspricht und welche Vor- und Nachteile diese Techniken haben. Zudem werden Messergebnisse dargestellt, welche die Dauer eines Handovers und den Gesprächsablauf verdeutlichen.

1. EINLEITUNG

Bei Behörden und Organisationen mit Sicherheitsaufgaben (BOS) wird die Analogfunktechnik für die Kommunikation verwendet. Bei dieser Technik wird das Sprachsignal auf ein Trägersignal aufmoduliert und über Funk auf einer reservierten und zugeteilten Frequenz übertragen und ausgegeben. Der größte Vorteil ist, dass die Technik sehr einfach im Verhältnis zum neuen Digitalfunk ist. Aufgrund dieser simpleren Technik gibt es aber auch Einschränkungen. So ist es beim Analogfunk nicht möglich die Daten zu verschlüsseln oder über bestimmte Funknetzgrenzen hinweg zu kommunizieren, denn beim Analogfunk gibt es meist pro Landkreis ein Gebiet, dem eine gewisse Frequenz zugeteilt wird. Diese Frequenz stellt dann den Kommunikationskanal dar, auf dem die Organisation in diesem Landkreis kommuniziert. Es ist daher nicht möglich, über diesen Kanal mit einer Organisation in einem anderen Landkreis zu kommunizieren. Aufgrund dieser und weiterer Nachteile, welche die Kommunikation über Regierungsgrenzen erschwert, wurde 1995 der TETRA-Standard standardisiert. Im Kapitel 3 wird daher auf die schon angesprochenen Vor- und Nachteile auch im Bezug zur TETRA Digitalfunktechnik eingegangen. Das darauf folgende Kapitel 4 behandelt TETRA und gibt Einblicke in die Funktionsweise. Im Kapitel 5 wird die Umsetzung des TETRA Standards auf das deutsche Digitalfunknetz der BOS dargestellt. Danach wird im Kapitel 6 auf Messungen, welche mittels zwei TETRA-Mobiles und einer Messsoftware von Rohde und Schwarz[1] durchgeführt wurde, eingegangen.

2. VERWANDTE ARBEITEN

Diese Arbeit bezieht sich hauptsächlich auf das Buch „Aufbau und Technik des digitalen BOS-Funks“ von Christof Linde[9] und auf die Dissertation Dimensionierung und Leistungsbewerung von TETRA-Bündelfunksystemen von Peter Sievering[10]. Zudem wurden genauere Details zum TETRA Standard aus dem Buch „Digital Mobile Communication and the TETRA System“[8] entnommen. Dieses Buch gibt einen umfangreicheren Einblick in den TETRA Standard. [9] gibt dagegen einen Einblick in den Digitalfunk aus Sicht der BOS und betrachtet somit eher eine abstrakte Sichtweise von TETRA.

3. DIGITALFUNK IM VERGLEICH ZUM ANALOGFUNK

Beim Analogfunk werden mehrere Funknetze für die verschiedenen BOS benötigt. Ein Grund dafür ist zum Beispiel, dass die Feuerwehr nicht die Gespräche der Polizei mithören kann oder deren Kanäle belegt. Durch die Analogfunktechnik sind aber meist die Funknetze auch auf Landkreisgrenzen beschränkt. Dies führt dazu, dass die Kommunikation von einem zum anderen Landkreis nicht einfach ist. Der Digitalfunk bietet dagegen allen BOS ein einheitliches, flächendeckendes Netz. Dadurch haben alle BOS in Deutschland neue Kommunikationsmöglichkeiten, welche die Zusammenarbeit vereinfachen und beschleunigen. Bei Großveranstaltungen oder Großschadenslagen können durch TETRA dynamische Gruppen erstellt werden, welche zum Beispiel Einheiten der Polizei und Feuerwehr auf die gleiche Kommunikations-Gruppen legt.[5]

Des Weiteren bietet der TETRA-Standard Abhörsicherheit, indem die Datenübertragung zwischen Endgerät und Basisstation (Luftschnittstelle) verschlüsselt wird. Zudem gibt es noch eine Ende-zu-Ende-Verschlüsselung, welche die Datenübertragung auch auf der restlichen Netzinfrastruktur schützt. Beim Analogfunk kann dagegen die Kommunikation, zum Beispiel der Polizei, mit der richtigen Frequenzwahl abgehört werden.[5]

Ein sehr wichtiger Punkt ist auch die Verbesserung der Sprach- und Empfangsqualität. Durch die Digitalisierung können Hintergrundgeräusche herausgefiltert werden und dies führt somit zu einer verbesserten Empfangsqualität. Beim Analogfunk werden keine Hintergrundgeräusche herausgefiltert und somit sind Gesprächspartner, welche in der nähe von Maschinen wie zum Beispiel einem Löschfahrzeug stehen, sehr schwer zu verstehen.[5]

[1]Elektronikkonzern mit Sitz in München (www.rohde-schwarz.de).

doi: 10.2313/NET-2012-04-1_6

Da durch die Bündelfunktechnik[2] mehrere Gruppen die gleiche Frequenz verwenden können, besteht eine höhere Frequenzökonomie. Somit sind die Funkkanäle auch nur dann belegt, wenn sie tatsächlich genutzt werden. Mit „belegt“ ist hier die feste Zuordnung von Organisationen zu Frequenzbereichen gemeint. Dabei kann dieser Frequenzbereich zu einer bestimmten Zeit nicht verwendet werden, allerdings kann auch keine andere Organisation auf diese Frequenzen zugreifen, da diese fest zugeordnet sind. Beim Analogfunk werden im Moment parallel bis zu sechs voneinander unabhängige analoge Funknetze (Polizei, Bundespolizei, Feuerwehr, Rettungsdienst, THW, Katastrophenschutz und Zoll), jeweils im 2- und im 4-Meter-Bereich[3] betrieben[1]. Zudem beansprucht dort jeder Kanal permanent einen Frequenzbereich. Daher kommt es bei Großschadenslagen auch regelmäßig zu Überlastungen. Alle Feuerwehren des Landkreises München haben zum Beispiel einen Kanal, auf dem diese mit der Leitstelle und untereinander kommunizieren können. Kommt es nun in Garching und Unterschleißheim zu einem Einsatz, bekommt jede der beiden Feuerwehren die Kommunikation der anderen mit und stören sich somit gegenseitig. Im schlimmsten Fall muss eine Feuerwehr mit der Nachforderung von Einsatzkräften warten, bis der andere sein Gespräch beendet hat. Mit TETRA könnte die Leitstelle beide auf verschiedene Kommunikations-Gruppen legen.[5]

Zudem bietet TETRA auch die Möglichkeit der Einzelkommunikation, welche es ermöglicht gezielt mit einem einzelnen Teilnehmer ein Funkgespräch zu führen. Beim Analogfunk können immer nur die Geräte erreicht werden, welche im gleichen Funknetz auf der gewählten Frequenz liegen. Durch die Wahl dieser Frequenz können somit auch immer beliebig viele Teilnehmer das Gespräch mitverfolgen. Somit ist eine Einzelkommunikation derzeit über Analogfunk nicht möglich.[5]

Eine sehr nützliche Funktion des Digitalfunks ist die Notruftaste. Durch das Drücken dieser Taste an den Endgeräten wird unter anderem ein Rufaufbau mit höchster Priorität zur Leitstelle veranlasst, welcher, falls das Funkgerät auch einen GPS-Empfänger hat, die aktuelle Position übermittelt und somit die Suche der verunglückten Einsatzkraft beschleunigt.[5]

4. TETRA

Das Kommunikationsnetz der BOS wurde 1995 aufgrund der europäischen Vereinheitlichung standardisiert. Dies geschah durch eine Projektgruppe des Europäischen Instituts für Telekommunikationsnormen (ETSI), welche ein einheitliches Funknetz für die europäischen Sicherheitsbehörden definieren sollte. Das Ergebnis war der dem Digitalfunknetz zugrunde liegende Standard TETRA. TETRA stand zu Beginn für Trans European Trunked Radio. Allerdings wurde dieser Begriff später aus kommerziellen Gründen in Terrestrial Trunked Radio geändert. Durch diese Bündelfunktechnik ist es möglich vielen Teilnehmern bzw. Gruppen gemeinsam ein Kanalbündel zur Verfügung zu stellen. Es besteht somit eine logische Zuordnung zu Gesprächsgruppen, die nicht über Frequenzen definiert werden müssen. Der TETRA Standard wird nicht nur von Behörden verwendet sondern auch von Industrie- und Nahverkehrsbetrieben, wie zum Beispiel den Stadtwerke München(SWM).

4.1 TopLevel-Beschreibung

Im Folgenden wird TETRA aus TopLevel Sicht beschrieben. Dabei werden die zwei Betriebsarten, Trunked Mode Operation und Direct Mode Operation, und der funktionale Aufbau des digitalen Mobilfunknetzes TETRA dargestellt.

4.1.1 Trunked Mode Operation (TMO)

Die normale Betriebsart ist TMO, bei der die Mobilgeräte die Netzinfrastruktur nutzen, um eine Kommunikation mit anderen Funkgeräten aufzubauen. Zwei Modi, die im Trunked Mode zur Verfügung stehen, sind der TETRA Voice + Date (TETRA V+D) Mode und Packet Data Optimized Mode (TETRA PDO). Allerdings wird bei den BOS bisher nur TETRA V+D verwendet, welcher der gleichzeitigen Übertragung von Sprache und Daten dient.[9]

TETRA V+D bietet kanalvermittelte Sprach- und Datendienste sowie einen Kurzdatendienst für Status- und Textnachrichten sowie einen Paketdatendienst für IP.[10]

Die Sprachdienste haben folgende Verbindungsarten:
Der **TMO-Gruppenruf** dient zur Kommunikation zwischen mehr als zwei Gesprächspartern und ist somit eine Punkt-zu-Multi-Punkt Verbindung. Im analogen Sprechfunk entspricht der Gruppenruf der hauptsächlichen Kommunikation. Daher ist diese Verbindungsart eine der Wichtigsten. Der Gruppenruf erfolgt im Halb-Duplex Modus. Dabei kann nur ein Gruppenmitglied senden und alle anderen Gruppenmitglieder empfangen auf dem gleichen Kanal. Durch die Nutzung des gleichen Downlink-Kanals ist dies auch eine effiziente Nutzung der Funkressourcen. Dieses Leistungsmerkmal besitzt GSM zum Beispiel nicht und ist daher einer der Gründe, warum man nicht GSM anstelle von TETRA verwendet.[9]

Beim **TMO-Einzelruf** wird eine Punkt-zu-Punkt Verbindung zwischen zwei Mobilteilnehmern aufgebaut. Dabei können beide Teilnehmer gleichzeitig empfangen und senden. Somit erfolgt die Kommunikation im Vollduplexmodus, welcher jeweils einen Uplink- und Downlink-Zeitschlitz benötigt. Der Kommunikationsaufbau entspricht dabei einem normalen Telefongespräch, bei dem man hier die Aliasnummer des Gesprächspartners wählt und dieser das Gespräch mit einem Knopfdruck annehmen muss.[9]

Des Weiteren gibt es im TMO Betrieb auch einen **Telefonruf**. Dieser ist vergleichbar mit dem TMO-Einzelruf mit der Erweiterung, dass man eine Verbindung in ein anderes Telekommunikationsnetz aufbauen kann.[9]

Beim Mobilfunknetz TETRA gibt es allerdings noch Prioritäten auf die Kommunikation. So können Gespräche, welche höhere Prioritäten haben, abhängig von Gruppen und Gerä-

[2]Der Bündelfunk ist eine Mobilfunkanwendung für Sprach- oder Datenübertragung mit einer oder mehreren Versorgungszellen. In jeder Zelle sind mehrere Übertragungskanäle verfügbar, von denen einer dynamisch zugewiesen wird, wenn ein Verbindungswunsch signalisiert wird. Durch diese dynamische Kanalzuweisung und durch die Bündelung mehrerer Kanäle werden eine sehr effiziente Frequenzausnutzung und eine hohe Verfügbarkeit gewährleistet.[6]

[3]Bei der Feuerwehr spricht man von der Wellenlänge statt Frequenz. Daher entspricht Fahrzeugfunk, welcher im 4-Meter-Bereich ist, dem 80-MHz-Bereich.

doi: 10.2313/NET-2012-04-1_6

ten, andere Gespräche mit niedrigeren Prioritäten verdrängen, falls alle verfügbaren Zeitschlitze ausgelastet sind. Der **Notruf** hat immer die höchste Priorität.[9]

4.1.2 Direct Mode Operation (DMO)

Bei DMO wird die Netzinfrastruktur nicht benötigt. Durch DMO ist es somit möglich bei einer Überlastung des Netzes noch in einem Gebiet zu kommunizieren. Des Weiteren kann man so auch unabhängige Gesprächsgruppen erstellen, welche das Netz nicht belasten. Um den Direct Mode verwenden zu können, müssen alle Gesprächsteilnehmer ihr Funkgeräte manuell auf Direct Mode umschalten. Da nun die Frequenz und Zeitschlitzwahl nicht mehr durch die Netzinfrastruktur vorgegeben werden kann, müssen alle den gleichen Kanal selbst wählen. Die Kommunikation kann hier nur im Halb-Duplex Modus durchgeführt werden, da der DMO-Master prüft, ob auf der gewählten Frequenz jemand sendet. Falls dies nicht der Fall ist, kann er senden.[9]

Der **DMO-Einzelruf** entspricht dem **DMO-Gruppenruf** mit einem statt mehreren Empfängern. Bei beiden kann nur über Halb-Duplex kommuniziert werden.[9]

Bei besonderen Ereignissen kann es sinnvoll sein eine DMO-Gruppe einzusetzen, obwohl das TMO-Netz zur Verfügung steht. Hierbei ist es gerade für Führungskräfte von Vorteil, wenn sie gleichzeitig im TMO und DMO Betrieb kommunizieren können. Daher gibt es Geräte, welche die **Dual Watch** Betriebsart unterstützen, bei der sie periodisch eine DMO-Frequenz und den Organisationskanal einer Basisstation überwachen.[9]

Beim Einsatz von Handfunkgeräten ist durch die Baugröße die Reichweite deutlich geringer als bei Fahrzeugfunkgeräten. Durch **DMO-Repeater** kann aber die Reichweite vergrößert werden. DMO-Repeater nehmen dabei das Signal vom Sender auf und senden dieses auf einem anderen Kanal oder Zeitschlitz weiter. Als DMO-Repeater können insbesondere Fahrzeugfunkgeräte eingesetzt werden, da diese aufgrund ihrer Bauart auch eine größere Sendeleistung als Handfunkgeräte haben.[9]

Ein weitere Betriebsart ist die **DMO-Gateway**. Falls ein Handfunkgerät beim Betrieb aufgrund der Sendeleistung nicht in der Lage wäre eine Basisstation zu erreichen, aber es ein Fahrzeugfunkgeräte in der Nähe gibt, welches eine Verbindung zur Basistation hat, besteht noch die Möglichkeit über DMO eine Verbindung zu einem Fahrzeugfunkgerät aufzubauen, welches als DMO-Gateway dient und aufgrund der größeren Sendeleistung die Basisstation erreicht.[9]

4.1.3 Funktionaler Aufbau des TETRA-Systems

Der Aufbau des TETRA-Systems ist ähnlich zu GSM. Es besteht aus einer Mobile Station(MS), Line Station (LS) sowie einer Switching and Management Infrastructure(SwMI).[10]

Die Mobile Station verfügt über eine eindeutige Kennung, die TETRA Subscriber Identity (TSI) (vgl. Abb. 1), welche aus dem Mobile Country Code (MCC), Mobile Network Code (MNC) und der Short Subscriber Identity (SSI) besteht. Der MCC beinhaltet die Länderkennung (262 für Deutschland). Der MNC bezeichnet das entsprechende TETRA-Netz, zum Beispiel 1001 für BDBOS. Die SSI steht für den Teilnehmer. Wenn man eine Verbindung zu einem bestimmten Funkgerät im Heimnetz aufbaut, wird nur die SSI als Adresse verwendet. Zudem gibt es noch für jede Mobile Station die TETRA Equipment Identity (TEI), welche gerätespezifisch ist. Diese Nummer wird vom Betreiber vergeben.[10]

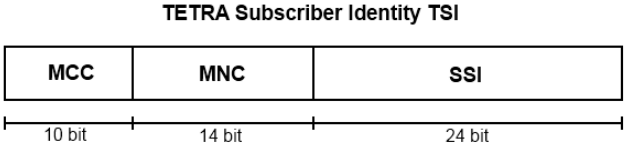


Abbildung 1: TETRA Subscriber Identity

Die Line Station bietet die gleichen Funktionen und Dienste wie die Mobile Station, ist allerdings über ISDN mit der Netzinfrastruktur verbunden. Die Leitstelle ist zum Beispiel eine Line Station. Diese hat auch den Zugang zu Datenbanken der Verwaltungsinfrastruktur.[10]

Die Switching and Management Infrastructure beinhaltet Basisstationen, Vermittlungsstellen (auch Main Switching Center (MSC) genannt), Transitvermittlungstellen und das Netzverwaltungszentrum (NMC), welche die Kommunikation zwischen Mobile Stationen und Line Stations herstellen und überwachen. Des Weiteren übernimmt das SwMI Aufgaben, welche unter anderem das Zuteilen von Kanälen und das Vermitteln von Verbindungen sind. Die Authentifizierung wird durch die SwMI durchgeführt und verfügt somit auch über die Home Data Base (HDB) und Visited Data Base (VDB).[10]

Die HDB enthält alle administrativen Informationen über jeden Mobilteilnehmer, der im Netz registriert ist. Die VDB enthält eine Teilmenge der HDB Daten, welche für die Rufüberwachung und das interne Routing im jeweils überwachten Bereich notwendig sind.[9]

Wie die HDB und VDB in den SwMI Komponenten verteilt sind, ist nicht Bestandteil des TETRA-Standards.[8]

4.2 LowLevel-Beschreibung

In diesem Abschnitt wird TETRA durch die unteren drei OSI-Schichten beschrieben. In Tabelle 1 sind die wichtigsten Parameter von TETRA aufgelistet.

4.2.1 Bitübertragungsschicht

Die Bitübertragungsschicht ist bei TETRA V+D zum einen für die Burstbildung verantwortlich. Bei der Burstbildung wird ein Burst auf eine Trägerfrequenz aufmoduliert. Der TETRA-Burst ist somit der Inhalt eines definierten Zeitschlitzes[9]. Das hierbei verwendete Modulationsverfahren ist $\pi/4$ Differential Quaternary Phase-Shift Keying (DQPSK), welches eine Modulationsbitrate von 36kBit/s hat.[10]

Bei TETRA gibt es verschiedene Basisstrukturen für einen Burst. Diese dienen zum einen der Datenübertragung aber auch der Sychronisierung und Änderung der Sendeleistung. Ein normaler Burst besteht aus zwei Blöcken mit je 216 Bit Nutzdaten, womit der Burst einem Zeitschlitz des Frames entspricht.[10]

doi: 10.2313/NET-2012-04-1_6

Tabelle 1: Technische Daten von TETRA

Parameter	Wert
Frequenzen (MHz)	UL:380-390 DL:390-400 UL:410-420 DL:420-430 UL:450-460 DL:460-470 UL:870-888 DL:915-933 (UL:Uplink, DL:Downlink)
Frequenzen BOS (MHz)	UL:380-385 DL:390-395 (UL:Uplink, DL:Downlink)
Kanalbandbreite	25 kHz
Duplexabstand	10 MHz
Rufaufbauzeit	<300 ms
Modulationsverfahren	$\pi/4$ Differential Quaternary Phase Shift Keying (DQPSK)
Modulationsbitrate	36 kbit/s
Mehrfachzugriff	TDMA mit 4 Zeitschlitzen pro Träger
Sprachcodec	Algebraic Code Excited Linear Predictive (ACELP) (4,56 kbit/s)
Nutzdatenrate (kbit/s)	hoch geschützt: 9,6 geschützt: 19,2 ungeschützt: 28,8

Zudem sind Modulation und Demodulation, Frequenz- und Symbolsynchronisation und Auswahl der Frequenzbänder und Sendeleistung Aufgaben dieser Schicht. Die Symbolsynchronisation wird durch eine Trainingssequenz erreicht, die in allen Bursts enthalten ist. Somit ist das Erkennen von Grenzen der Bursts möglich. In Europa sind die Frequenzbänder 410-420 MHz, 450-470 MHz, 870-876 MHz und 915-921 MHz reserviert. Der Duplexabstand zwischen Uplink- und Downlink-Trägerfrequenzen soll 10 MHz betragen. In Deutschland ist der Bereich 380-385 MHz als Uplink und 390-395 MHz als Downlink der BOS zugeteilt. Dabei entspricht ein Kanal der Bandbreite von 25 kHz und hat einen Schutzabstand von 25 kHz. Zudem messen Mobile Station den Empfangspegel des verwendeten Frequenzkanals um anhand der Werte einen Zellwechsel einzuleiten.[10]

4.2.2 Sicherungsschicht

Diese Schicht ist in die Teilschichten Medium Access Control (MAC) und Logical Link Control (LLC) aufgeteilt. Dabei gelangen die Daten von der Bitübertragungsschicht zuerst zur MAC-Teilschicht und dann über die LLC-Teilschicht zur Vermittlungsschicht.

Die MAC-Teilschicht umfasst die Funktionen der Kanalcodierung und Kanalzugriffssteuerung.[10]

TETRA verwendet als Multiplexverfahren Time Division Multiple Access (TDMA), welches jede Trägerfrequenz in vier Zeitschlitze der Größe von 510 Bit einteilt. Ein TDMA-Rahmen hat daher eine Größe von 255 Byte. Achtzehn Rahmen werden zu einem Multirahmen zusammengefasst, der 4590 Byte groß ist. Ein Hyperrahmen besteht aus 60 aufeinander folgenden Multirahmen, mit der Größe von ca. 269 kB. In Abbildung 2 ist die Rahmenstruktur dargestellt.[10]

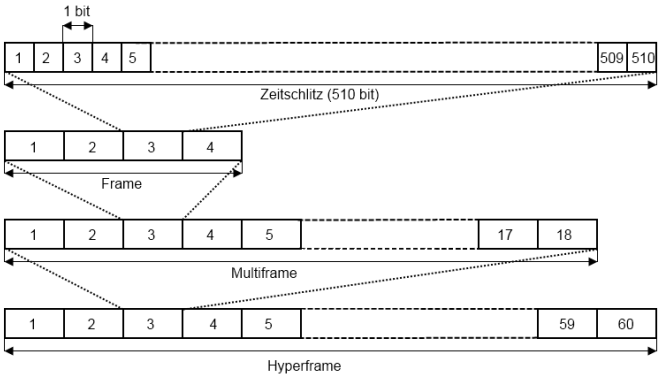


Abbildung 2: TETRA-Rahmenstruktur

Die LLC-Teilschicht ist für die Datenübertragung, Übertragungswiederholung und Segmentierung zuständig und stellt so der Vermittlungsschicht fehlerfreie Daten zur Verfügung.[8]

4.2.3 Vermittlungsschicht

Diese stellt Funktionen zur Verfügung um Verbindungen zwischen mobilen Teilnehmern, Teilnehmern an Line Stations und Teilnehmern in anderen Netzen aufzubauen.[10]

Der Layer 3 ist in die Teilschichten Mobile Link Entity (MLE), Circuit Mode Control Entity (CMCE), Mobilitätsmanagement (MM) und Subnetwork Dependent Convergence Protocol (SNDCP) unterteilt.[10]

- Die Aufgabe der Mobile Link Entity-Teilschicht ist das Herstellen und Aufrechterhalten der Funkverbindung zwischen Mobile- und Basisstation. Daher entscheidet auch die MLE-Teilschicht wann und ob sie die Funkzelle wechselt. Die MLE besteht wiederum aus Komponenten, welche für die Überwachung der Funkkanalqualität, das Scannen und Bewerten von Nachbarzellen, das Durchführen von Zellwechselprozeduren und die Auswahl der LLC Dienste verantwortlich sind.[10]
- Die Circuit Mode Control Entity ist verantwortlich für die Rufsteuerung und Kurzdatendienste.[10]
- Das Mobilitätsmanagement (MM) besitzt die Aufgabe die Zuordnung von Mobile Stationen zu Aufenthaltsbereichen aktuell zu halten.[10]
- Das SNDCP ist für die Einbindung von IP-Teilnetzen verantwortlich. So wird in dieser Schicht die Zuordnung von einer PDP-Adresse (z.B. IP-Adresse) zu einer Individual TETRA Subscriber Identity (ITSI) kontrolliert und verwaltet.[10]

doi: 10.2313/NET-2012-04-1_6

4.2.4 Handover

In einem Netz aus vielen Funkzellen müssen die Funkverbindungen über die Dauer des gesamten Gesprächs auch beim Wechseln der Zelle aufrecht erhalten werden. Den Vorgang, bei dem ein Gespräch auf eine Frequenz in der nächsten Zelle weitergegeben wird, bezeichnet man als Handover. Dieser Handover kommt zustande, wenn die Kommunikationsbedingung einer benachbarte Zelle besser als die der eigenen ist. Wie die Kommunikationsbedingung der benachbarten Zellen sind, wird durch das Überprüfen der Control Channels[4] der jeweiligen Basistationen festgestellt.[9]

Beim Handover gibt es unterschiedliche Arten aufgrund der Netztopologie:
1) Die Umschaltung kann zwischen unterschiedlichen Basisstationen erfolgen, welche an der gleichen Vermittlungsstelle (MSC) angeschlossen sind. In diesem Fall kann die Vermittlungstelle den Vorgang selbst verarbeiten.
2) Die Umschaltung kann auch zwischen unterschiedlichen Basisstationen, welche an unterschiedlichen Vermittlungsstellen (MSC) angeschlossen sind erfolgen. Beide Vermittlungsstellen sind allerdings an das selbe Transit Main Switching Center (Transit MSC) angeschlossen.
3) Zudem kann die Umschaltung zwischen unterschiedlichen Basistationen erfolgen, welche an unterschiedlichen Main Switching Centers angeschlossen sind. Diese Form wird auch als externes Handover bezeichnet.[9]

Welche Nachrichten bei einem Handover auf Layer 3 Ebene entstehen, wird in Abschnitt 6 behandelt.

4.3 Sicherheit

TETRA hat drei Mechanismen um Daten sicher zu übertragen. Eine Sicherheitsvorkehrung ist die Luftschnittstellenverschlüsselung. Diese Verschlüsselung basiert auf den einzelnen Funkzellen. Beim TMO-Einzelruf handelt jeder Teilnehmer mit der Basisstation den jeweiligen Kommunikationsschlüssel einzeln aus. Dagegen verwenden beim TMO-Gruppenruf alle Gruppenmitglieder den gleichen Kommunikationsschlüssel. Beim DMO Betrieb werden weitere statische Kommunikationsschlüsseln eingesetzt, welche in den Geräten gespeichert sind.[9]

Des Weiteren gibt es noch eine Ende-zu-Ende Verschlüsselung, welche auch die Daten zwischen Basisstation und Leitstelle verschlüsselt. Im TETRA-Netz der BOS wird diese Sicherheit durch ein asymmetrisches Kryptosystem realisiert, bei der jede Mobile Station eine BSI-Kryptokarte benötigt. Diese entspricht einer intelligenten SIM Karte und enthält einen speziell für die Verschlüsselung optimierten Prozessor mit zugehörigem Datenspeicher[9]. Die BSI-Kryptokarte beinhaltet aber auch noch die operativ-taktische Adresse, durch die jedes Funkgerät eindeutig zuordenbar wird.[7]

5. TETRA BEI BOS

Das TETRA-Netz wird bei der BDBOS seit 2008 ausgebaut. Nach den Angaben von BDBOS soll die flächendeckende Infrastruktur voraussichtlich bis Ende 2012 aufgebaut sein.[3]

[4]Der Control Channel ist ein unidirektionaler Steuerkanal zwischen der Basis Station und den Mobile Stations. Über diesen Kanal werden Netzinformationen von der Basis Station übertragen.[9]

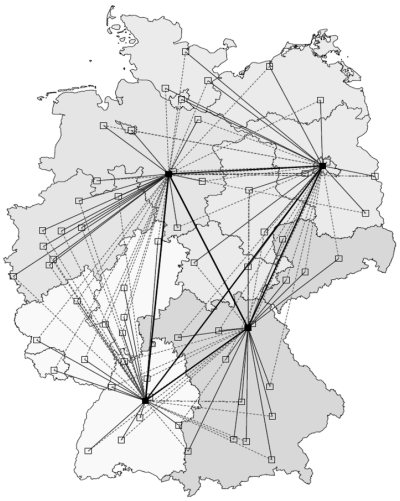


Abbildung 3: Funktionaler Aufbau des digitalen BOS-Netzes[2]

Zum funktionalen Aufbau des digitalen BOS Netzes (vgl. Abb. 3). Dieses besteht aus 45 Netzabschnitten. Es gibt in diesen Netzabschnitten 64 Vermittlungsstellen (MSC), welche mit den einzelnen Basisstationen über Kabel oder Richtfunk verbunden sind. Die Vermittlungsstellen sind wiederum mit zwei Transitvermittlungsstellen verbunden. Hiervon gibt es vier, welche vollvermascht vernetzt sind[2]. Somit sind die Vermittlungsstellen mit mehreren Transitvermittlungsstellen verbunden und beim Ausfall eines Knoten ist die Kommunikation noch über einen weiteren gewährleistet.[9]

Die gesamte Steuerung des Netzes erfolgt über zwei Netzverwaltungszentren (NMC). Diese befinden sich in Berlin und Hannover, um bei einem Störungsfall sich gegenseitig ersetzen zu können.[4]

Die Leitstellen sind über Kabel mit zwei unabhängigen Vermittlungsstellen verbunden. Durch diese Redundanz wird die Leitstelle bei einem Ausfall einer Leitung nicht funktionslos.[9]

6. TETRA MESSUNGEN

Aufgrund mehrerer Mitteilungen von Feuerwehrangehörigen, die besagen, dass man beim neuen BOS-Funk lange warten muss bis man sprechen kann oder beim Zellwechsel für mehrere Sekunden die Verbindung ausfällt, wurden einige Messungen durchgeführt, um festzustellen wie lang ein Verbindungsaufbau dauert und ein Handover an Zeit benötigt.

Diese Messungen konnte mit der Unterstützung von Rohde und Schwarz durchgeführt werden. Dabei kam die Messsoftware ROMES 4.63 und zwei Sepura TETRA-Mobiles zum Einsatz. Die Mobiles waren über USB mit dem PC verbunden, auf dem die ROMES lief. Über die SAIL Schnittstelle der Sepura Mobiles wurden dann die entsprechenden Daten ausgelesen. Eigentlich sollte die Messungen im Netz der BOS durchgeführt werden. Dafür wären allerdings Mobiles notwendig gewesen, welche im BOS-Netz eingebucht sind. Rohde und Schwarz hat allerdings nur zwei Mobiles, welche im Netz der Stadtwerke München eingebucht sind. Beim digitalen

doi: 10.2313/NET-2012-04-1_6

BOS-Netz können sich die folgenden Zeiten somit unterscheiden, da dort eine andere und größere Netzinfrastruktur vorliegt. Bei der Messung befanden sich beide Mobiles im TMO Betrieb. In dieser Betriebsart hat das Mobile 2 mehrere Einzelrufe an Mobile 1 durchgeführt. Das Mobile 2 hat im Abstand von 60 Sekunden Mobile 1 angerufen und die Verbindung für 30 Sekunden gehalten.

Aus den Layer 3 Nachrichten in Abbildung 4 kann man den Verbindungsaufbau und Abbau erkennen. Die Zeitangaben der Layer 3 Nachrichten sind aus den Messsprotokollen entnommen. Die Messsoftware ROMES protokolliert mit den Layer 3 Nachrichten auch den Zeitpunkt.

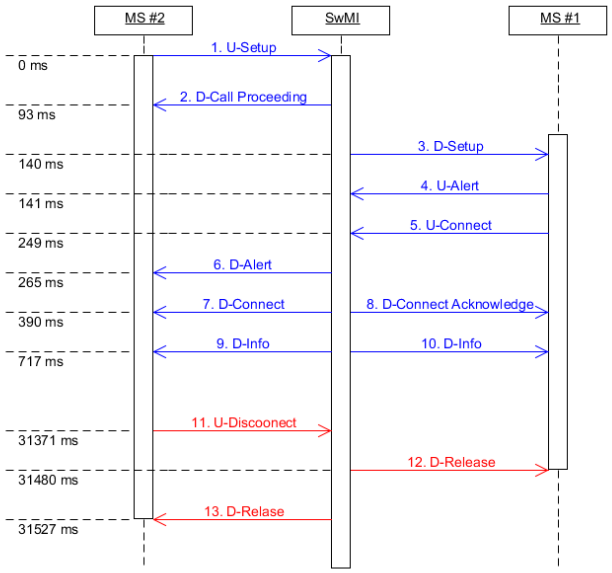


Abbildung 4: TETRA Call

Bei dieser Verbindung wird als erstes die Protocol Data Unit (PDU) U-Setup von MS2 an die SwMI geschickt um zu signalisieren, dass das MS2 eine Verbindung aufbauen will. Daraufhin erhält das MS2 vom SwMI als Bestätigung die PDU D-Call Proceeding, dass der Gesprächsaufbau fortgesetzt wird. Danach sendet die SwMI an das MS1 die PDU U-Setup, um diesem mitzuteilen, dass es von MS2 angerufen wird. Als Bestätigung erhält die SwMI das U-Alert. Sobald das MS1 bereit für die Verbindung ist, sendet dies U-Connect an die SwMI. Das MS2 erhält nun auch die PDU D-Alert als Bestätigung, dass das MS1 verfügbar ist. Nun sendet die SwMI die PDU D-Connect an MS2 und D-Connect Acknowledge an MS1 als Bestätigung, dass die Verbindung aufgebaut ist. Mit der PDU D-Info werden noch allgemeine Daten an die Mobiles geschickt. Nach den 30 Sekunden beendet MS2 mit der PDU U-Disconnect das Gespräch. Die SwMI schickt darauf an MS1 und MS2 D-Release um zu signalisieren, dass das Gespräch beendet wurde.

Bei einem Handover entstehen die in Abbildung 5 gezeigten Layer 3 Nachrichten.

Wenn das Mobile eine neue Zelle ausgewählt hat, schickt dies die PDU U-Prepare an die SwMI mit der gewählten Zelle als

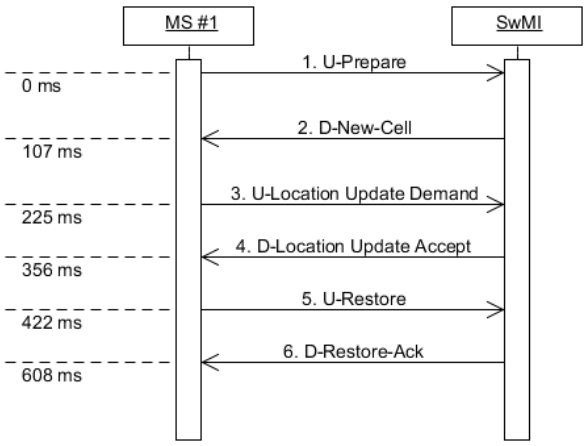


Abbildung 5: TETRA Handover

Information. Das SwMI antwortet darauf, indem es die PDU D-NEW-CELL schickt, um zu bestätigen, dass das Mobile in die gewünschte Zelle wechseln kann. Als nächstes sendet das Mobile die PDU U-Location Update Demand, um bei der SwMI eine Aktualisierung der Registrierungen zu veranlassen. Die SwMI schickt, sobald das Update durchgeführt wurde, die PDU D-Location Update Acceptet. Danach sendet das Mobile noch U-Restore, welches der SwMI signalisiert, dass das Mobile noch Informationen aufgrund des Zellwechsels aktualisiert. Die SwMI bestätigt dies mit D-Restore-Ack.

Aus diesen zwei Auschnitten kann man aber nicht generell auf den zeitlichen Abblauf eines Gesprächs schließen. Somit können sich je nach Auslastung des Netzes oder auch des Mobiles diese Zeiten sehr unterscheiden. Auch die Frage, wie lang bei dem Handover aus Abbildung 5 die Tonübertragung unterbrochen war, konnte man mit diesen Layer 3 Nachrichten nicht zeigen. Vermutlich fällt die Übertragung der Sprache ab der PDU D-NEW-CELL bis U-RESTORE aus, was 315 ms[5] entsprechen würde. Allerdings kann man hier gut erkennen wie die Kommunikation in TETRA abläuft.

7. ZUSAMMENFASSUNG UND ZUKUNFT VON TETRA

Zum einen wurde immer wieder bemängelt, dass der TETRA Standard schon wieder veraltet ist und die Übertragungsraten zu niedrig sind. Dies ist aber nur in einem gewissen Maß korrekt. Im Moment steht bei einer verschlüsselten Verbindung eine Übertragungsrate von 9,6 kbit/s zur Verfügung, was im Vergleich zu UMTS, mit zum Teil verfügbaren 7,2 Mbit/s, sehr gering ist. Allerdings sind diese 9,6 kbit/s nicht dem TETRA Standard an sich verschuldet sondern aufgrund der BDBOS Auslegung. So kann mit einem anderen Modulationsverfahren und größeren Kanalbandbreiten auch eine höhere Übertragungsrate erreicht werden. Zudem gibt es auch schon Erweiterungen von TETRA. Wie den TETRA Enhanced Data Service, mit dem eine Steigerung der Übertragungsrate mit bis zu 300 kBits/s möglich ist. Falls diese Datenrate nicht ausreicht, besteht immer noch

[5] time(„D-NEW-CELL“) - time(„U-RESTORE“) = 107 ms - 422 ms = 315 ms

doi: 10.2313/NET-2012-04-1_6

die Möglichkeit LTE als Zusatz für die Datenübertragung zu verwenden. Diese höheren Datenraten sind vor allem auch nur für Datenübertragungen notwendig. Für die normale Sprachkommunikation reicht die Übertragungsrate von 9,6 kbit/s durch den Sprachcodec aus.

Die Frage stellt sich nun vielleicht, warum man nicht gleich einen Mobilfunkstandard wie LTE verwendet. Gründe warum man dies nicht machen sollte, sind zum einen, dass LTE, GSM usw. die Gruppenruffunktion und Verschlüsselung nicht unterstützen. Es wurde aber auch über die Nutzung des GSM Netzes nachgedacht. Allerdings hat ein eigenes Funknetz immer den großen Vorteil, dass dies auch noch funktionsfähig ist, wenn bereits andere Netze aufgrund von Überlastung ausgefallen sind. So ist es vor allem bei Massenveranstaltungen sinnvoll, ein eigenes Kommunikationsnetz zu haben, da dort öfters die Mobilfunknetze GSM bzw. UMTS überlastet sind.

Im großen und ganzen bringt die Einführung vom Digitalfunk den BOS eindeutig mehr Vorteile und wird somit der neue Kommunikationsstandard der BOS in Deutschland.

8. LITERATUR

[1] Bayerisches Staatsministerium des Innern (StMI). Ablösung Analogfunk BOS-Digitalfunk - Einführung in Bayern. `http://www.stmi.bayern.de/sicherheit/digitalfunk/einfuehrung/detail/17342/`. 21.01.2011.

[2] Bundesanstalt für den Digitalfunk der Behörden und Organisationen mit Sicherheitsaufgaben. Digitalfunk BOS Vorsorgemaßnahmen zur Energieversorgung. PowerPoint.

[3] Bundesanstalt für den Digitalfunk der Behörden und Organisationen mit Sicherheitsaufgaben (BDBOS). Fortschrittsanzeiger: Der Aufbau des BOS-Digitalfunknetzes. `http://www.bdbos.bund.de/cln_090/nn_1358092/DE/Bundesanstalt/Projekt__Digitalfunk/Netzaufbau__Roll__out/Fortschrittsanzeiger/fortschrittsanzeiger__roll__out__node.html?__nnn=true`. 21.01.2011.

[4] Bundesanstalt für den Digitalfunk der Behörden und Organisationen mit Sicherheitsaufgaben (BDBOS). Netzverwaltungszentren des BOS-Digitalfunknetzes in Betrieb. `http://www.bdbos.bund.de/cln_090/nn_1358092/DE/Bundesanstalt/Projekt__Digitalfunk/Netzaufbau__Roll__out/Netzverwaltungszentren/netzverwaltungszentren__node.html?__nnn=true`. 21.01.2011.

[5] Bundesanstalt für den Digitalfunk der Behörden und Organisationen mit Sicherheitsaufgaben (BDBOS). Warum eigentlich Digitalfunk BOS? `http://www.bdbos.bund.de/cln_090/nn_421176/DE/Bundesanstalt/Projekt__Digitalfunk/Vorteile__des__Digitalfunks__BOS/vorteile__node.html?__nnn=true`. 21.01.2011.

[6] Bundesnetzagentur. Bündelfunk. `http://www.bundesnetzagentur.de/DE/Sachgebiete/Telekommunikation/RegulierungTelekommunikation/Frequenzordnung/FrequenzzuteilungAntraege/Buendelfunk/Buendelfunk_node.html`. 21.01.2011.

[7] J. T. D. Christof Linde. *Digitalfunk*. ecomed SICHERHEIT, 2011.

[8] J. I. John Dunlop, Demessie Girma. *Digital Mobile Communications and the TETRA System*. John Wiley and Sons Ltd, 1999.

[9] C. Linde. *Aufbau und Technik des digitalen BOS-Funks*. Franzis Verlag, 2008.

[10] P. Sievering. Dimensionierung und Leistungsbewertung von TETRA-Bündelfunksystemen. Disseration, Dezember 2004.

doi: 10.2313/NET-2012-04-1_6

Minimum-Cost Multicast over Coded Packet Networks

Jan Schalkamp
Advisor: Stephan M. Günther
Seminar: Innovative internet technologies and mobile communications WS11/12
Chair for Network Architecture and Services
Department of Computer Science, Technische Universität München
Email: jan.schalkamp@in.tum.de

ABSTRACT

Streaming data to different receivers from a single source is a very common scenario. In general this would be achieved by using single source multicast, for which finding the most cost-efficient paths is very hard as it has to be computed in a centralized manner and is only affordable for little networks since it is not solvable in polynomial time. Although today's networks use e.g. heuristics to quickly find sub-optimal solutions to transmit data from one source to multiple receivers, it would be nice to find even better solutions in less time. However, changing the problem a little bit might help. Although routers are currently only able to forward and copy packets, there is no reason to limit them to only those functionalities. Having a network of routers capable of network coding – meaning being able to apply an arbitrary causal function on multiple incoming packets, resulting in one outgoing packet – can change this. This work gives an introduction to finding optimal cost solutions for multicast in a coded packet network, and shows that finding these solutions is possible in a decentralized manner resulting in only polynomial effort to calculate.

Keywords

Coded Packet Network, Multicast, Network Coding, Lagrangian Dual Problem, Optimization Problem, Minimum Cost Flow Problem

1. INTRODUCTION

Sending data from a single source to multiple sinks is an important task of today's networks. This, for example, is the case for video streaming services like YouTube, as the amount of traffic from video streaming is a crucial part of their expenses. Therefore it is vitally to reduce these cost as good as possible. There is a multitude of ways to achieve the minimal cost, which depend mostly on the characteristics of the network being used. If routers were only able to forward packets, the weighted shortest paths to each sink are the most cost-efficient way of delivering the data. These paths can easily be found by using the widely known Dijsktra algorithm. But as today's routers are also capable of copying packets and forwarding them to different receivers, the task of optimizing cost becomes more difficult. In general multicast would be used to transmit data from a single source to multiple sinks. However, finding the minimal cost in this kind of network becomes very difficult as calculating a minimum spanning tree is $\mathcal{NP}$-complete, which leads to a not affordable cost of computation. As there is no reason to limit routers to only being able to copy and forward packets, coded packet networks were introduced by Ahlswede et al [3]. In these networks, routers are able to combine multiple incoming packets by an arbitrary function to only one outgoing packet. This work, based on Desmund Lun's paper *Minimum-Cost Multicast Over Coded Packet Networks*, shows that this new characteristic of a network yields a way of minimizing the cost for multicast in a decentralized manner, which is also solvable in polynomial time. Therefore, we first take a look at the ways to deliver data from a single source to multiple sinks in different networks. Secondly a general optimization problem for minimizing costs in routed and coded packet networks are formulated. As solving these optimization problems would be out of the scope of this work, it is only shown that a specific kind of cost function leads to a way of solving the optimization problem in a decentralized manner.

2. COMPARING WAYS TO DELIVER INFORMATION TO MULTIPLE SINKS

As this work focuses on optimizing the cost for multicast in coded packet networks it is of great help to compare these networks with routed networks first. Therefore we represent a network by a graph $\mathcal{G} = (\mathcal{N}, \mathcal{A})$, that is directed, weighted. $\mathcal{N}$ determines the set of Nodes, which would for example be routers in a real world application. The arcs between nodes are depicted by the set $\mathcal{A}$. Furthermore, we write an arc of set $\mathcal{A}$ as tuple (i, j) with $i, j \in \mathcal{N}$ as the arc's start, respectively end nodes. To model a network, we also need some kind of capacity c_{ij} which is the maximum rate at which packets can be transmitted over an arc (i, j) and cost for that transmission, which will be denoted by the arc's positive weight a_{ij}.

2.1 Dedicated unicast in routed networks

As we wish to transmit data from a single source $s \in \mathcal{N}$ to multiple sink nodes $t \in \mathcal{T}$ with $\mathcal{T} \subset \mathcal{N} \backslash \{s\}$, it is the easiest approach to send the data to the sinks one by one. Therefore, one unicast connection per sink node is established from the source to the sink node and the same information is transmitted $|\mathcal{T}|$ times. To transmit with the lowest cost the shortest path to each node is evaluated, for example by using the Dijsktra algorithm [6]. This results in a shortest path tree, giving the source the necessary information to send it's data to each sink via the lowest cost path per sink node. This would be the minimum-cost way to distribute data in a network only able to establish unicast connections. But as we will see in the forthcoming section 2.2, distributing the same information to multiple sink nodes by unicast is

doi: 10.2313/NET-2012-04-1_7

not the best way to do it. An example of the unicast approach can be seen in Figure 1. Here the shortest paths from the source Q to the sinks S_1 and S_2 are $(Q,1),(1,S_1)$ and $(Q,2),(2,S_2)$. Each path has costs of $2\cdot(5+2)$, resulting in a total cost of 28.

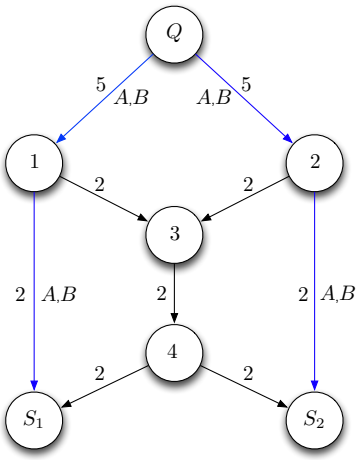


Figure 1: Unicast example

2.2 Multicast in routed networks

In networks with nodes being capable of not only forwarding packets but also copying and directing these packets to multiple nodes, multicast is a way to distribute information to multiple sinks. Due to nodes being able to copy packets it is no longer necessary to send each packet $|\mathcal{T}|$ times as can be seen in Figure 2. Here, the information A and B is only sent once to node 4 at costs of $2\cdot(5+2+2)=18$ and afterwards node 4 copies the received information and sends it to each sink at costs of $2\cdot(2\cdot 2)=8$, resulting in total costs of 26. So, as one can see in comparison to the unicast's cost of 28, multicast is a better solution than unicast. As the optimal distribution path in routed networks is a Steiner tree, the total costs of transmitting information can only be as high as using the shortest path to each sink – just like the unicast solution does. This would be the case, if the costs for arcs $(1,3)$ and $(2,3)$ in our example were increased to 4. Calculating such a Steiner tree in polynomial time is not possible, which leads to a problem for big networks [4]. Nevertheless multicast without the use of Steiner trees is used in todays networks, as even sub-optimal solutions, calculated with the help of heuristics and specialized multicast protocols, are better than using unicast. In section 2.3 we see, that it is possible to find even lower cost solutions in polynomial time in a special kind of networks.

2.3 Multicast in coded packet networks

Just like in section 2.2 and real world networks, our nodes are able to forward, copy and direct copies to multiple nodes. But there is no reason to limit nodes to only that functionality. In coded packet networks, nodes are additionally able to use an arbitrary causal function on multiple incoming packets. This can be used to combine two packets , e.g. by using bitwise XOR $(A\oplus B)$, in order to reduce the total amount of packets needed to be transmitted through the network. As in the example of Figure 3, node 3 combines the incoming

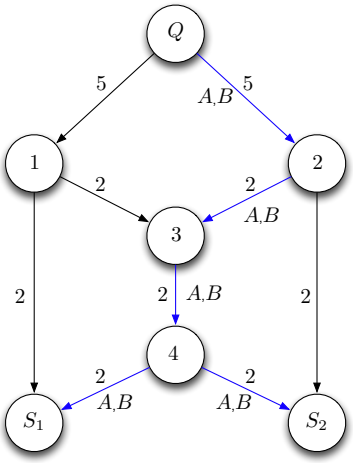


Figure 2: Multicast example

packets A and B to the new packet $A\oplus B$ and sends it via 4 to both sink nodes. Now, the packet $A\oplus B$ is useless for each sink, as they are not able to derive both original packets A and B from it. Therefore sink S_1 receives packet A and sink S_2 receives packet B, as this enables both sinks to derive the other packet by using XOR on $A\oplus B$ and A, respectively B. Now each arc is only used once and the total cost are 24. Just like in routed networks, the multicast depends greatly on the network's structure. Increasing the costs of arcs $(1,3)$ and $(2,3)$ again to 4, the optimal solution would be once more a unicast connection per sink. The advantage of using multicast in coded packet networks is not only the possibly lower cost of sending packets, but also the time to calculate those solutions. As finding the Steiner tree in a routed network was not solvable in polynomial time, network coding enables us to find optimal solutions for multicast in polynomial time, making it interesting for real world networks.

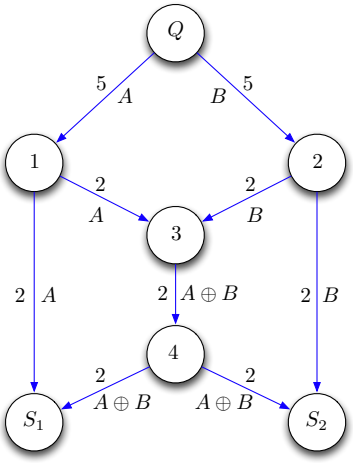


Figure 3: Multicast in coded packet networks example

doi: 10.2313/NET-2012-04-1_7

3. MULTICAST AS OPTIMIZATION PROBLEM

On the way of finding a solution to the problem of minimizing cost in a coded packet network, we first have to state our goal as an optimization problem in order to provide suitable algorithms capable of solving it. As we already have declared the capacity of an arc as c_{ij} with $(i,j) \in \mathcal{A}$, we denote z_{ij} as actual rate at which packets are injected into the arc. As the costs of a transmission in a network emerge from even those rates z_{ij} – because only used arcs lead to costs – we can state the monotonous increasing cost function depending on the vector $\vec{z}$:

$$f(\vec{z}) : \mathbb{R}^{|\mathcal{A}|} \to \mathbb{R}^+ := [0, \infty). \tag{1}$$

Minimizing this function as it stands would lead to a not usable solution, as we haven't modeled our network correctly. Therefore, we develop some constraints, which help us to state our conditions for the network. As above stated, z_{ij} is the rate at which packets are injected into arc (i,j) and c_{ij} is the capacity of arc (i,j). Therefore we state our first constraint, the *capacity constraint*:

$$c_{ij} \geq z_{ij}, \forall (i,j) \in \mathcal{A}. \tag{2}$$

Sending some information over an arc $(i,j) \in \mathcal{A}$ does not necessarily mean that this information is new and also does not show which sink it is provided for. Therefore, we introduce $x_{ij}^{(t)}$ as the rate of new information intended for sink $t \in \mathcal{T}$. As the rate of new information for a single sink can not be greater than the overall rate of information at arc (i,j) and the rate of information must not be negative, as it would mean that information could be destroyed or lost, we state the *coupling constraint*:

$$z_{ij} \geq x_{ij}^{(t)} \geq 0, \forall (i,j) \in \mathcal{A}, \forall t \in \mathcal{T}. \tag{3}$$

Our third constraint describes the fact, that our network only has one source node $s \in \mathcal{N}$ that is providing information and multiple sink nodes $t \in \mathcal{T}$ that are consuming even that information. No inner node is allowed to either produce or consume information. This constraint is the *flow conservation constraint*:

$$\sum_{\{j|(i,j)\in\mathcal{A}\}} x_{ij}^{(t)} - \sum_{\{j|(j,i)\in\mathcal{A}\}} x_{ji}^{(t)} = \sigma_i^{(t)}, \forall i \in \mathcal{N}, \forall t \in \mathcal{T} \tag{4}$$

where

$$\sigma_i^{(t)} := \begin{cases} R, & \text{if } i = s \\ -R, & \text{if } i \in \mathcal{T} \\ 0, & \text{otherwise.} \end{cases}$$

R means the real, positive rate at which packets are transmitted to the sink nodes. For dedicated unicast this is easy to see, as the source node is producing information, the inner nodes forward every received information without tampering with it and sink nodes consume the information. In routed networks using multicast it is not that easy to see, as inner nodes are permitted to copy information and therefore a single packet may be of use for more than one sink. Therefore, the amount of incoming and outgoing packets does not have to be the same. In Figure 2, node 4 is an example for that. It receives packets A and B once but transmits them twice. To see that our third constraint (4) is still valid, we have to look at the actual information being hold by the packets received by node 4. As both A and B hold information for sinks S_1 and S_2, the received and the transmitted amount of information is equal. To make it easier to see, we take a closer look at the example of Figure 2 in numbers (only a look at node 4): Each packet's information rate from node 3 to 4 is $x_{3,4}^{(S_1)} = x_{3,4}^{(S_2)} = \frac{1}{2}$ since both sinks S_1 and S_2 are equally interested in packet A's as well as in packet B's information. Each packet provides half of the overall information each sink is interested in. Therefore, the incoming information is $2 \cdot (\frac{1}{2} + \frac{1}{2}) = 2$. The outgoing information to both sinks S_i is $x_{4,S_i}^{(S_i)} = \frac{1}{2}$ for packet A as well as B, which results in an outgoing amount of information of $2 \cdot (2 \cdot \frac{1}{2}) = 2$ as well. This holds also for the case of a coded packet network.

These three constraints are enough to tackle the problem of minimizing the cost in a multicast network. However, there are some differences between the optimization problem in routed and coded packet networks. This is dealt with in the two forthcoming sub chapters.

3.1 Optimizing multicast in a routed network

As we are now able to formulate the optimization problem, we first have a look at minimizing the cost for multicast in routed networks with one source node and multiple sink nodes. Of course, we still assume that the rates of new information $x_{ij}^{(t)}$ for sink t are greater than or equal to 0. Also, constraint (4), introduced as *flow conservation constraint*, and the *capacity constraint* (2), shall be valid in this network. Since we want to minimize a given cost function like (1) our optimization problem reads as follows:

$$\begin{aligned}
&\text{minimize } f(\vec{z}) \\
&\text{subject to } c_{ij} \geq z_{ij} && \forall (i,j) \in \mathcal{A}, \\
&z_{ij} \geq \sum_{t\in\mathcal{T}} x_{ij}^{(t)}, x_{ij}^{(t)} \geq 0 && \forall (i,j) \in \mathcal{A}\ ,\ \forall t \in \mathcal{T}, \\
&\sum_{\{i|(i,j)\in\mathcal{A}\}} x_{ij}^{(t)} - \sum_{\{i|(j,i)\in\mathcal{A}\}} x_{ji}^{(t)} = \sigma_i^{(t)} && \forall i \in \mathcal{N}\ ,\ \forall t \in \mathcal{T}.
\end{aligned}$$

Note, that the second constraint has changed a little bit in comparison to (3), as we now explicitly have a routed network in which packets cannot be coded. Now, as we have upper and lower bounds for the optimal vector $\vec{z}$, namely the capacity (2) and coupling constraint (3), we see that our optimal vector $\vec{z}$ lies within the positive orthant[1] of a bounded polyhedron. Being monotonically increasing, our cost function $f(\vec{z})$ can only be minimized by minimizing each of it's components z_{ij}. This, on the other hand, is reached when the coupling constraint $z_{ij} \geq \sum_{t\in\mathcal{T}} x_{ij}^{(t)}$, $x_{ij}^{(t)} \geq 0, \forall (i,j) \in \mathcal{A}, \forall t \in \mathcal{T}$ holds with equality.

3.2 Optimizing multicast in a coded packet network

We now consider a coded packet network. Nodes are able to apply arbitrary causal functions on multiple incoming packets like $A \oplus B$. The problem looks nearly the same as in a routed network, but since nodes are able to use network

[1] An n-dimensional vector lies within the positive orthant, iff all it's components are positive.

doi: 10.2313/NET-2012-04-1_7

coding, the coupling constraint changes accordingly:

$$\begin{aligned}
&\text{minimize } f(\vec{z}) && (5)\\
&\text{subject to } c_{ij} \geq z_{ij} && \forall (i,j) \in \mathcal{A},\\
&z_{ij} \geq x_{ij}^{(t)} \geq 0 && \forall (i,j) \in \mathcal{A}\ ,\ \forall t \in \mathcal{T},\\
&\sum_{\{i|(i,j)\in\mathcal{A}\}} x_{ij}^{(t)} - \sum_{\{i|(j,i)\in\mathcal{A}\}} x_{ji}^{(t)} = \sigma_i^{(t)} && \forall i \in \mathcal{N}\ ,\ \forall t \in \mathcal{T}.
\end{aligned}$$

As nodes are now able to code packets, they also become able to send the same amount of information with less packets. As Figure 3 shows, arc (3,4) only has to be used once but still carries the same amount of information for sinks S_1 and S_2 (assuming S_1 received A and S_2 received B already). The coupling constraint is less restrictive. $\vec{z}$ is now optimal iff the following equality holds for every component of $\vec{z}$ [2]:

$$z_{ij} = \max_{t\in\mathcal{T}}\{x_{ij}^{(t)}\} = ||x_{ij}^{(t)}||_{\infty,(t)},\ \forall (i,j) \in \mathcal{A},\ \forall t \in \mathcal{T}. \quad (6)$$

As we have not specified the cost function $f(\vec{z})$ yet, the next section deals with solving the optimization problem for a specific class of cost functions.

4. SOLVING THE OPTIMIZATION PROBLEM IN CODED PACKET NETWORKS

This section deals with a specific class of cost functions. We now only consider linear, separable cost functions, and separable constraints for each arc $(i,j) \in \mathcal{A}$. Separable constraints mean in this case that each arc's capacity is subject to a separate constraint, bounding it to a positive value independent from the other arcs' bounded capacities. Linear means, that the cost of transmitting data over arc $(i,j) \in \mathcal{A}$ grows linear with the amount of data transmitted. For instance, this is the case if the cost represent monetary cost, such as a fixed value per kilobyte. We also demand that the cost of transmitting data is non-negative. These constraints lead to a cost function $f(\vec{z})$ looking as follows:

$$\begin{aligned}
f(\vec{z}) &:= \sum_{(i,j)\in\mathcal{A}} a_{ij} z_{ij} && (7)\\
&\text{with } a_{ij} \geq 0\ \forall (i,j) \in \mathcal{A}.
\end{aligned}$$

In the forthcoming sub sections, we formulate the optimization problem in two different ways, as a different formulation of the problem may lead to a different way of solving the optimization.

4.1 A first approach to stating the optimization problem

In this approach, the optimization problem looks nearly the same as the previous stated general optimization problem (5). We just rewrote the coupling constraint, so that z_{ij} is no longer bounded explicitly above. However, it is still bounded above implicitly by the characteristics of a coded packet network (6), that limits z_{ij} by the maximum of the arc's flows $x_{ij}^{(t)}$.

$$\begin{aligned}
&\text{minimize } \sum_{(i,j)\in\mathcal{A}} a_{ij} z_{ij} && (8)\\
&\text{subject to } c_{ij} \geq z_{ij} \geq 0 && \forall (i,j) \in \mathcal{A},\\
&z_{ij} \geq x_{ij}^{(t)} && \forall (i,j) \in \mathcal{A}\ ,\ \forall t \in \mathcal{T},\\
&\sum_{\{i|(i,j)\in\mathcal{A}\}} x_{ij}^{(t)} - \sum_{\{i|(j,i)\in\mathcal{A}\}} x_{ji}^{(t)} = \sigma_i^{(t)} && \forall i \in \mathcal{N}\ ,\ \forall t \in \mathcal{T}.
\end{aligned}$$

Although solving this problem would lead to the optimal solution, we are not satisfied with this form of the problem, as it cannot be solved in a distributed manner. This might seem a little strange as it does look a lot like a normal minimum-cost flow problem for which a variety of algorithms exist [8]. Unfortunately our flow conservation constraint (4) is referring to the flows $x_{ij}^{(t)}$ and not to the vector $\vec{z}$. This is a result of network coding, as flows can now share bandwidth instead of competing for it. Hence, as it stands, this problem has a major disadvantage: it requires full knowledge of the network and is only solvable in a centralized manner.

4.2 The optimization as dual problem

Dualizing a problem has the benefit of reducing the amount of constraints. Those constraints are not be lost, but are dualized into the problem's function. We therefore have to formulate the Lagrangian of the primal problem (8) to obtain the dual function, which will be maximized in order to obtain the dual problem. However, it is possible that optimizing the dual problem may lead to a different solution than the primal problem, wherefore the equality of this case will be shown.

Dualizing the coupling constraint $z_{ij} \geq x_{ij}^{(t)}$ forms the following Lagrangian:

$$\begin{aligned}
\mathcal{L}(\vec{x},\vec{z},\vec{\lambda}) &= f(\vec{z}) + \sum_{(i,j)\in\mathcal{A}}\sum_{t\in\mathcal{T}} \lambda_{ij}^{(t)}(x_{ij}^{(t)} - z_{ij})\\
&= \sum_{(i,j)\in\mathcal{A}} a_{ij} z_{ij} + \sum_{(i,j)\in\mathcal{A}}\sum_{t\in\mathcal{T}} \lambda_{ij}^{(t)}(x_{ij}^{(t)} - z_{ij})\\
&= \sum_{(i,j)\in\mathcal{A}} a_{ij} z_{ij} + \sum_{(i,j)\in\mathcal{A}}\sum_{t\in\mathcal{T}} (\lambda_{ij}^{(t)} x_{ij}^{(t)} - \lambda_{ij}^{(t)} z_{ij})\\
&= \sum_{(i,j)\in\mathcal{A}} a_{ij} z_{ij} - \sum_{(i,j)\in\mathcal{A}}\sum_{t\in\mathcal{T}} \lambda_{ij}^{(t)} z_{ij} + \sum_{(i,j)\in\mathcal{A}}\sum_{t\in\mathcal{T}} \lambda_{ij}^{(t)} x_{ij}^{(t)}\\
&= \sum_{(i,j)\in\mathcal{A}} z_{ij} \cdot \left(a_{ij} - \sum_{t\in\mathcal{T}} \lambda_{ij}^{(t)}\right) + \sum_{(i,j)\in\mathcal{A}}\sum_{t\in\mathcal{T}} \lambda_{ij}^{(t)} x_{ij}^{(t)}.
\end{aligned}$$

Since we dualized the coupling constraint, it is no longer an explicit constraint of our optimization problem. The remaining constraints, the capacity and flow conservation constraint are transformed into a set $\mathcal{X}$ that contains all vectors $\vec{x}$ fulfilling even these constraints. Towards stating the dual problem, the dual function $\Theta(\vec{\lambda})$ has to be found first by minimizing the Lagrangian $\mathcal{L}$ with respect to $\lambda_{ij}^{(t)} \geq 0\ \forall (i,j) \in \mathcal{A}, \forall t \in \mathcal{T}$. As z_{ij} is no longer bounded above, it is allowed to take any value in order to minimize the Lagrangian $\mathcal{L}$, wherefore

$$\min_{\vec{x}\in\mathcal{X},\vec{z}} \sum_{(i,j)\in\mathcal{A}} z_{ij} \cdot \left(a_{ij} - \sum_{t\in\mathcal{T}} \lambda_{ij}^{(t)}\right) + \sum_{(i,j)\in\mathcal{A}}\sum_{t\in\mathcal{T}} \lambda_{ij}^{(t)} x_{ij}^{(t)} \quad (9)$$

doi: 10.2313/NET-2012-04-1_7

evaluates to $-\infty$, if $a_{ij} - \sum_{t\in\mathcal{T}} \lambda_{ij}^{(t)}$ is either greater or less than 0. In case $\sum_{t\in\mathcal{T}} \lambda_{ij}^{(t)} = a_{ij}$ it is exactly zero and the minimized Lagrangian (9) evaluates to $\min_{\vec{x}\in\mathcal{X}} \sum_{(i,j)\in\mathcal{A}} \sum_{t\in\mathcal{T}} \lambda_{ij}^{(t)} x_{ij}^{(t)}$.

Therefore the dual function $\Theta(\vec{\lambda})$ looks as follows:

$$\Theta(\vec{\lambda}) := \inf_{\vec{x}\in\mathcal{X},\vec{z}} \mathcal{L}(\vec{x},\vec{z},\vec{\lambda})$$

$$\inf_{\vec{x}\in\mathcal{X},\vec{z}} \mathcal{L}(\vec{x},\vec{z},\vec{\lambda}) = \begin{cases} \min_{\vec{x}\in\mathcal{X}} \sum_{(i,j)\in\mathcal{A}} \sum_{t\in\mathcal{T}} \lambda_{ij}^{(t)} x_{ij}^{(t)}, & \text{if } \sum_{t\in\mathcal{T}} \lambda_{ij}^{(t)} = a_{ij}, \\ -\infty, & \text{otherwise.} \end{cases}$$

Although $-\infty$ would be a correct mathematical solution, it holds no meaning in reality, wherefore we eliminate that possible solution by adding a constraint concerning the Lagrangian multipliers λ, leading to the dual problem:

$$\begin{aligned} &\text{maximize} && \Theta(\vec{\lambda}) && (10)\\ &\text{with} && \Theta(\vec{\lambda}) := \min_{\vec{x}\in\mathcal{X}} \sum_{(i,j)\in\mathcal{A}} \sum_{t\in\mathcal{T}} \lambda_{ij}^{(t)} x_{ij}^{(t)}, \\ &\text{subject to} && \sum_{t\in\mathcal{T}} \lambda_{ij}^{(t)} = a_{ij}, \lambda_{ij}^{(t)} \geq 0 \ \forall (i,j)\in\mathcal{A}, \forall t\in\mathcal{T}. \end{aligned}$$

To see that the optimal solution to the dual problem is equal to the primal problem's (8) optimal solution, we have to show that strong duality[2] holds. This can be done by rewriting the primal problem's dualized constraint (the coupling constraint) as constraint functions:

$$h_{ij}^{(t)}(z_{ij}) := x_{ij}^{(t)} - z_{ij} \leq 0.$$

It is obvious that each $h_{ij}^{(t)}$ is less than or equal to zero, because z_{ij} is equal to or greater than $x_{ij}^{(t)}$ (6). Slater's condition[3] holds in case all constraint functions are less than or equal to zero and at least one $h_{ij}^{(t)} < 0$ exists. Since each sink needs at least one raw packet to retrieve the information of the coded packet, it is guaranteed that such an $h_{ij}^{(t)}$ exists and strong duality is proven [7].

We rewrite our problem to make differences to the primal problem (8) more obvious:

$$\begin{aligned} &\text{maximize} && \Theta(\vec{\lambda}) \\ &\text{with} && \Theta(\vec{\lambda}) := \sum_{t\in\mathcal{T}} \zeta^{(t)}, && (11)\\ & && \zeta^{(t)} := \min_{\vec{x}^{(t)}\in\mathcal{X}^{(t)}} \sum_{(i,j)\in\mathcal{A}} \lambda_{ij}^{(t)} x_{ij}^{(t)} && (12)\\ &\text{subject to} && \sum_{t\in\mathcal{T}} \lambda_{ij}^{(t)} = a_{ij}, \lambda_{ij}^{(t)} \geq 0 \ \forall (i,j)\in\mathcal{A}, \forall t\in\mathcal{T}. \end{aligned}$$

Note, that the set $\mathcal{X}$ has now been separated by flows into multiple sets $\mathcal{X}^{(t)}$. Since we broke the coupling between the flows by dualizing the coupling constraint, we can now use distributed methods to solve the optimization problem. As mentioned in section 4.1, the primal problem looked a lot like a normal minimum-cost flow problem. By reformulating the dual problem (10), we defined a $\zeta^{(t)}$(12), which is in fact a standard minimum-cost flow problem and can therefore be solved e.g. by using the ϵ-relaxation method. The overall optimization (11) can then be done by using subgradient-optimization. Both techniques will not be covered at this point. Subgradient-optimization of even this optimization problem is to be found in [1], whereas the ϵ-relaxation is discussed in [5].

[2]Strong duality holds, iff the difference between the primal problem's and the dual problem's solution is 0. Here, (minimize $f(\vec{z})$) − (maximize $\Theta(\vec{\lambda})$) = 0.

[3]Slater's condition is a constraint qualification guaranteeing equality of primal and dual optimal solutions.

5. CONCLUSION

As we have seen by comparing the different methods for delivering data from a single source to multiple sinks, solving the optimization problem strongly depends on the network's capabilities. In case of a routed network with unicast capabilities, the optimization problem is even $\mathcal{NP}$-complete. Hence, we concentrated on minimizing the cost in coded packet networks wherefore we first had a look at how to state the optimization problem with respect to the three constraints (capacity, coupling and flow conservation) in general. Afterwards we specified a linear cost function in order to see, that the first approach of solving the problem would not have been solvable in polynomial time. However, by dualizing the coupling constraint, and abbreviating the capacity and flow conservation constraint by the set $\mathcal{X}$, we were able to obtain the dual problem by maximizing the Lagrangian and adding an additional constraint in order to exclude a not meaningful solution. It was also crucial to show that the dual problem's optimal solution was the same as the optimal primal problem's solution by showing that strong duality holds with help of Slater's condition. In the end, we rewrote the dual problem to see that a part of it is a standard minimum-cost flow problem for which a variety of algorithms exist. Having solved that particular subproblem, the whole problem became solvable by employing subgradient-optimization.

6. REFERENCES

[1] Desmond S. Lun, Niranjan Ratnakar, Muriel Medard, Ralf Koetter, David R. Karger, Tracey Ho, Ebad Ahmed and Fang Zhao: *Minimum-Cost Multicast Over Coded Packet Networks*, IEEE Transactions on information theory, Vol. 52 No.6, pp. 2608 - 2623, 2006.

[2] Stephan M. Günther: *Optimal Cost Multicast over Coded Packet Networks*, Optimization in Communication and Signal Processing, Ferienakademie Sarntal, 2009.

[3] Rudolf Ahlswede, Ning Cai, Shuo-Yen Rober Li and Raymond W. Yeung: *Network Information Flow*, IEEE Transactions on information theory, Vol 46., no. 4, pp. 1204 - 1216, 2000.

[4] F. Hwang, D.S. Richards and P. Winter: *The Steiner tree problem*, North Holland, 1992.

[5] D.P. Bertsekas and J.N. Tsitsiklis: *Parallel and Distributed Computation: Numerical Methods.*, Englewood Cliffs, NJ: Prentice Hall, 1989

[6] T.H. Cormen and C.E. Leiserson, R.L Rivest and C. Stein: *Introduction to algorithms*, MIT Press, Cambridge, Ma, 2001.

[7] Stephen Boyd and Lieven Vandenberghe: *Convex Optimization*, 6th edition, Cambridge University Press, 2008.

doi: 10.2313/NET-2012-04-1_7

[8] Ravindra K. Ahuja, Thomas L. Magnanti, James B. Orlin: *Network Flows: Theory, Algorithms, and Applications.*, Prentice-Hall, Inc., 1993.

doi: 10.2313/NET-2012-04-1_7

WebSockets: Spezifikation / Implementierung

Benjamin Ullrich
Betreuer: Philipp Fehre
Hauptseminar: Innovative Internettechnologien und Mobilkommunikation WS2011/12
Lehrstuhl Netzarchitekturen und Netzdienste
Fakultät für Informatik, Technische Universität München
Email: bu@elunic.com

KURZFASSUNG

Webseiten gehen heutzutage oft über die reine Darstellung statischer Hypertexte hinaus. Entwickler nutzen die Vorteile browserbasierter Anwendungen und schaffen interaktive Web-Applikationen mit dynamischen Inhalten, die vom Anwender unabhängig von Ort und Endgerät genutzt werden können. Das HTTP-Protokoll genügt den Anforderungen an die Kommunikation zwischen Client und Server dabei nur bedingt. Durch eine Vielzahl an notwendigen Serverabfragen erzeugt es einen großen Overhead beim Datenaustausch und verursacht unnötige zeitliche Verzögerungen.

Mit der Einführung von HTML5 soll auch das neue WebSocket-Protokoll in modernen Webbrowsern zum Einsatz kommen, das bidirektionale Verbindungen zwischen Webbrowser und Webserver ermöglicht und den notwendigen Netzwerktraffic minimieren soll. Im ersten Teil dieser Arbeit werden das JavaScript-Interface, technische Aspekte der Protokoll-Spezifikation und der Einsatzzweck von WebSockets untersucht. Im zweiten Teil wird die konkrete Implementierung von WebSockets aufgezeigt und die Fragestellung beantwortet, wie diese neue Technologie trotz eingeschränkter Browserkompatibilität schon heute eingesetzt werden kann.

Schlüsselworte

WebSocket Protokoll, HTML5, HTTP Header-Overhead, bidirektionale Webbrowser-Kommunikation, Comet

1. MOTIVATION

Der Einsatzzweck von Webseiten geht längst über den Abruf statischer Informationen hinaus. Technische Fortschritte der Webbrowser, wie leistungsstärkere JavaScript-Engines, eingehaltene browserübergreifende Konventionen und Bibliotheken wie jQuery ermöglichen die Entwicklung dynamischer Web-Applikationen. Beispiele wie Facebook oder Google-Docs zeigen, dass moderne Webapplikationen in vielen Bereichen auch als Alternative zu klassischen Desktop-Anwendungen eingesetzt werden können. Vorteile sind etwa:

- Keine clientseitige Installation der Anwendung notwendig
- Ortsunabhängige Verfügbarkeit der Services
- Client-Kompatibilität (verschiedene OS, Smartphones, Tablets...)
- Lizenzmodelle, die schwer oder gar nicht umgangen werden können

Nachteile sind dagegen:

- Notwendigkeit einer Internetverbindung
- Webbrowser-Kompatibilitätsprobleme durch Abweichungen von Standards (z.B. Internet Explorer 6)
- eingeschränkte Performance mit clientseitiger JavaScript-Programmierung
- eingeschränkter Zugriff auf Rechner-Ressourcen mit clientseitiger JavaScript-Programmierung im Gegensatz zu Desktop-Programmiersprachen (z.B. Grafikkarte, Netzwerksockets)

Nach wie vor stellen vor allem die Einschränkungen der JavaScript-Programmierung einen Nachteil gegenüber Desktopapplikationen dar. Mit der Einführung von HTML5 wurden daher neue Standards für Webbrowser definiert, die die Möglichkeiten der clientseitigen JavaScript-Programmierung erweitern.

Eine dieser Neuerungen sind “WebSockets”, deren Spezifikation in dieser Arbeit untersucht werden soll. Während Desktop-Entwickler Verbindungen zwischen Endgeräten aufbauen und über diese bidirektional kommunizieren können, beschränkt sich das HTTP-Protokoll auf die serverseitige Beantwortung clientseitiger Anfragen. Dies hat zur Folge, dass der Webbrowser für jede Aktualisierung von Daten eine neue Anfrage an den Server stellen muss. Anstatt je nach Bedarf Daten über eine offene Verbindung kommunizieren zu können, muss außerdem für jede Anfrage eine neue Verbindung aufgebaut werden. Dadurch entsteht ein Daten-Overhead, unnötige Rechenleistung wird beansprucht und die Aktualisierung der Daten wird unnötig verzögert.

Ein Beispiel hierfür ist die Realisierung eines Webseiten-Chats. Da die Client-Anwendung nicht weiß, wann neue Chat-Nachrichten verfügbar sind, muss sie regelmäßig neue Anfragen an den Server senden. Diese enthalten dann gegebenenfalls Informationen über neue Nachrichten. Ist das Abfrageintervall lang, erscheinen Nachrichten erst mit einiger Verzögerung beim Chatpartner. Ist das Abfrageintervall kurz, werden u.U. viele unnötige Anfragen gesendet, obwohl

doi: 10.2313/NET-2012-04-1_8

in der Zwischenzeit keine neuen Nachrichten gesendet wurden.

Das HTTP-Protokoll genügt den Anforderungen moderner Webapplikationen an eine effiziente Kommunikation zwischen Client und Server daher nur bedingt. Mit HTML 5 wurde das WebSocket Protokoll vorgestellt. Dieses ermöglicht Entwicklern von Web-Anwendungen das Öffnen von TCP-Verbindungen für einen bidirektionalen Datenaustausch mit einem Webserver. Bisherige Technologien, die zu diesem Zweck mehrere HTTP-Verbindungen öffnen mussten, sollen dadurch überflüssig gemacht werden.

Der erste Teil dieser Arbeit beschreibt das WebSocket Protokoll und zeigt, wie dieses eine sichere Verbindung zwischen Webbrowser und Server ermöglicht. In Abschnitt 3 wird darauf aufbauend eine vom World Wide Web Consortium (W3C) definierte Client-API[4] für WebSockets vorgestellt[1]. Anhand einer Beispielanwendung wird demonstriert, welche Auswirkungen der Einsatz von WebSockets als Alternative zu HTTP-Anfragen auf die Netzlast haben kann. Zum Abschluss der Arbeit werden Einschränkungen in der Kompatibilität mit gängigen Webbrowsern beschrieben und gezeigt, wie die Vorteile von WebSockets bereits heute genutzt werden können.

2. WEBSOCKET PROTOKOLL

Das WebSocket Protokoll wurde seit seiner Bekanntgabe kontinuierlich weiterentwickelt und im Dezember 2011 von der IETF als offizieller Internetstandard eingeführt [7]. Dieser Abschnitt beschreibt, welche Funktionsweisen die Spezifikation für Webbrowser und Server vorgibt, um eine sichere Verbindung zu gewährleisten.

2.1 Verbindungsaufbau

Bei der Initialisierung neuer WebSockets baut der Webbrowser eine TCP-Verbindung zu der angegebenen Server-Adresse auf. Zu Beginn wird ein Handshake durchgeführt. Mit diesem stellt der Browser sicher, dass der Server das WebSocket-Protokoll versteht und es werden Parameter der Verbindung spezifiziert.

Eine Herausforderung bei der Einführung von WebSockets war die einfache Integrierbarkeit in die etablierte Infrastruktur des Internets. Das WebSocket-Protokoll wurde daher als Upgrade des HTTP-Protokolls konzipiert und kann so ohne aufwändige Neukonfigurationen auch in bestehenden Web-Servern über Port 80 genutzt werden. Der Opening Handshake einer WebSocket-Verbindung muss daher ein valider HTTP-Request sein. Die IETF weist aber darauf hin, dass in zukünftigen Versionen des Protokolls auch ein einfacherer Handshake über einen dedizierten Port denkbar ist. [3]

Header 1 zeigt den WebSocket Request-Header. Wie bei HTTP ist die erste Zeile nach dem Request-Line Format aufgebaut, bestimmt also Methode, angefragte Ressource (Request-URI), Protokoll und Protokollversion. Die Methode ist dabei auf den Wert „GET“ festgelegt. Die zweite Zeile gibt den adressierten Host und optional einen Port an, falls die Verbindung nicht über Port 80 laufen soll. Über den Wert „websocket“ des Upgrade-Feldes in Kombination mit dem Wert „Upgrade“ des Connection-Feldes wird der Host aufgefordert, auf das WebSocket-Protokoll zu wechseln. [3]

[1] s. http://dev.w3.org/html5/websockets

Header 1 Opening Handshake: Request-Header [3]

```
GET /chat HTTP/1.1
Host: server.example.com
Upgrade: websocket
Connection: Upgrade
Origin: http://example.com
Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
Sec-WebSocket-Version: 13
Sec-WebSocket-Protocol: chat, superchat
```

Header 2 Opening Handshake: Response-Header [3]

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=
Sec-WebSocket-Protocol: chat
```

Daneben definiert das Protokoll die Verwendung einiger zusätzlicher Felder, die eine sichere Verbindung gewährleisten sollen oder mit denen Parameter der Verbindung spezifiziert werden können [3]:

Origin

Das Feld muss vom Webbrowser mitgeschickt werden, um vor unerlaubten Cross-Domain Anfragen in Webbrowsern zu schützen. Der Server kann durch die Auswertung des Felds also selbst entscheiden, von welchen Adressen er Anfragen zulassen möchte.

Sec-WebSocket-Key

Das Feld enthält einen zufälligen 16-Byte-Wert der base64-kodiert wurde.

Sec-WebSocket-Version

Die Version des Protokolls; nach aktueller Spezifikation muss das Feld den Wert „13“ besitzen.

Sec-WebSocket-Protocol (optional)

Das Feld zeigt an, welche Anwendungsschicht-Unterprotokolle vom Client unterstützt werden.

Sec-WebSocket-Extension (optional)

Protokoll-Erweiterungen, die vom Client unterstützt werden. Diese Angabe ermöglicht zukünftig die Integration von Erweiterungen des Protokolls, wie eine Frame-Komprimierung oder Multiplexing mehrerer WebSocket-Verbindungen[2], ohne sie für alle Anwendungsfälle zwingend vorauszusetzen.

Optional können analog zu HTTP weitere Felder zwischen Client und Server kommuniziert werden, wie beispielsweise Cookies.

[2] s. auch http://tools.ietf.org/html/draft-tamplin-hybi-google-mux-01

doi: 10.2313/NET-2012-04-1_8

Der korrekte Response-Header des Servers (vgl. Header 2) zeigt dem Client an, dass der Server das WebSocket-Protokoll versteht. Die erste Zeile ist nach dem HTTP Status-Line Format aufgebaut, enthält also Protokoll, Version und Status-Code. Ist der Status-Code ungleich „101“, so kann der Webbrowser die Antwort analog zu HTTP behandeln, um beispielsweise eine Autorisierung bei Code „401“ durchzuführen. [3]

Der Server muss den HTTP-Upgrade Prozess durch Angabe derselben Felder-Werte für „Connection“ und „Upgrade“ wie im Request-Header vervollständigen. Weitere durch das Protokoll bestimmte Felder sind außerdem [3]:

Sec-WebSocket-Accept

Mit dem Feld muss ein umgeformter Wert des im Request-Header empfangenen „Sec-WebSocket-Key“ übergeben werden. Der empfangene Wert muss hierzu mit der GUID „258EAFA5-E914-47DA-95CA-C5AB0DC85B11“ verknüpft werden. Der Server muss dann einen SHA-1 Hash des verketteten Wertes generieren, diesen base64-kodieren und das Ergebnis als Wert des Header-Feldes „Sec-WebSocket-Accept“ zurückgeben.

Damit kann der Webbrowser sicherstellen, dass der Server die Anfrage wirklich gelesen und verstanden hat. Sollte der Browser einen falschen Wert für Sec-WebSocket-Accept empfangen, muss er die Antwort laut Protokoll als serverseitige Ablehnung der Verbindung werten und darf daher keine zusätzlichen Frames senden.

Sec-WebSocket-Protocol (optional)

Wurden vom Client ein oder mehrere Unterprotokolle empfangen, so kann der Server maximal eines davon für die Verbindung auswählen. Durch Zurücksenden des Wertes im „Sec-WebSocket-Protocol“ wird dem Client die Wahl des Unterprotokolls bestätigt.

Sec-WebSocket-Extension (optional)

Wurden vom Client unterstützte Extensions empfangen, so kann der Server durch Zurücksenden einzelner oder aller Extensions deren Verwendung für die WebSocket-Verbindung bestätigen.

Optional können wiederum weitere Felder zwischen Server und Client kommuniziert werden, wie beispielsweise „Set-Cookie“ um das Cookie zu überschreiben.

2.2 Datenübertragung

Nach einem erfolgreichen Opening Handshake wird clientseitig das WebSocket-Event „onopen“ angestoßen und es können bidirektional Nachrichten ausgetauscht werden. Die Nachrichten können jeweils aus einem oder mehreren Frames bestehen. So wird einerseits ermöglicht, Nachrichten ohne vorheriges Puffern zu senden. Andererseits kann ein Teilen der Nachrichten in mehrere kleinere Frames vorteilhaft für ein zukünftiges Multiplexing mehrerer Socket-Verbindungen sein.[3] Um die Kommunikation vor „Cache Poisoning“ zu schützen, müssen alle vom Client an den Server gesendeten Frames maskiert [5] werden.

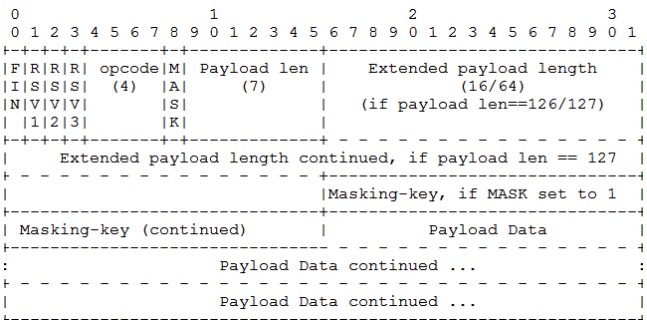


Abbildung 1: Framing-Modell

Eine Grundphilosophie des WebSocket-Protokolls ist es, den Overhead beim Datenaustausch möglichst gering zu halten. Daran orientiert sich auch das Framing der einzelnen Pakete. Abbildung 1 zeigt den Aufbau eines WebSocket-Frames. Das „FIN“-Bit gibt an, ob es sich um das letzte Frame der Nachricht handelt. Die folgenden drei Bits sind für Protokoll-Erweiterungen reserviert. Bits vier bis sieben beinhalten den OP-Code, der angibt, wie der Nachrichteninhalt interpretiert werden muss. Beispielsweise sind Textframes durch den OP-Code „0x1“ gekennzeichnet. Das achte Bit gibt an, ob das Frame maskiert ist. Die nächsten Bits geben die Länge das Nachrichteninhalts an. Je nach Umfang des Nachrichtinhalts werden für diese Angabe 7, 23 oder 71 Bits verwendet. Falls das Maskierungsbit gesetzt wurde, müssen die nächsten 32 Bit den Maskierungskey angeben. Alle weiteren Bits werden für den Nachrichteninhalt verwendet. [3]

Je nach Nachrichtenlänge und Maskierung werden pro Frame also 2 - 14 Bytes an Headern benötigt.

2.3 Verbindungsabbau

Um die Verbindung zu schließen, wird ein Closing-Handshake durchgeführt (vgl. Abbildung 2). Die Seite, die die Verbindung schließen möchte, muss dazu ein Frame mit dem Befehlscode „0x8“ senden. Optional kann im Body der Nachricht der Grund für das Schließen enthalten sein. Die ersten zwei Bytes des Bodys müssen dann einen Statuscode für den Grund enthalten. [3]

Empfängt eine Seite einen Close-Frame, ohne zuvor selbst einen Close-Frame gesendet zu haben, muss sie einen Close-Frame als Antwort senden. Nachdem eine Seite sowohl einen Close-Frame empfangen als auch gesendet hat, muss sie die TCP-Verbindung trennen. Durch dieses Vorgehen stellt es auch kein Problem dar, falls beide Seiten zur selben Zeit die Verbindung trennen möchten: Da beide Seiten parallel einen Close-Frame senden und kurz darauf den der jeweils anderen Seite empfangen, können sie die Verbindung direkt trennen. [3]

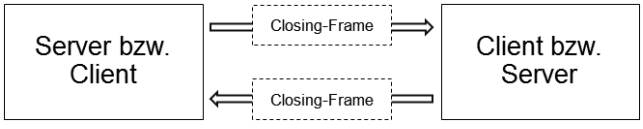


Abbildung 2: Closing Handshake

doi: 10.2313/NET-2012-04-1_8

3. WEBSOCKET API

3.1 Das WebSocket Interface

Quelltext 1 zeigt das Interface eines WebSockets nach [4], das dem Anwendungsentwickler clientseitig zur Verfügung steht.

3.1.1 Constructor

Der Constructor eines WebSockets besitzt den Parameter „url" und den optionalen Parameter „protocols". Für die Adressierung des WebSocket-Servers wurden zwei URI-Schemen analog zu HTTP eingeführt [4]:

ws-URI: **ws://host[:port][/path][?query]**
und wss-URI (secure): **wss://host[:port][/path][?query]**

Mit dem zweiten Parameter „protocols" können ein oder mehrere bevorzugte Protokolle für die Kommunikation angegeben werden.

Quelltext 1 WebSocket Interface (vereinfacht) [4]

```
[Constructor(url, optional protocols)]

interface WebSocket : EventTarget {

        readonly attribute url;

        // ready state
        const CONNECTING = 0;
        const OPEN = 1;
        const CLOSING = 2;
        const CLOSED = 3;

        readonly attribute readyState;
        readonly attribute bufferedAmount;

        // networking
        attribute Function onopen;
        attribute Function onerror;
        attribute Function onclose;

        readonly attribute extensions;
        readonly attribute protocol;

        void close(optional code, optional reason);

        // messaging
        attribute Function onmessage;

        attribute binaryType;

        void send(data);
};
```

3.1.2 Methoden

Nach der Initialisierung stehen dem JavaScript-Entwickler die beiden Methoden „send" und „close" zur Verfügung [4]:

send(data:mixed) Durch den „send"-Aufruf werden die Daten des Parameters „data" an den Server gesendet. Mögliche Formate des „data"-Parameters sind:

String Text-Nachricht (Textframes)

ArrayBuffer Daten des ArrayBuffers (Binärframes)

Blob Rohdaten (Binärframes)

close(optional code, optional reason) Mit dem Aufruf der Methode „close" wird die TCP-Verbindung des WebSockets geschlossen.

Quelltext 2 MessageEvent Interface [1]

```
interface MessageEvent : Event {

        readonly attribute data;
        readonly attribute origin;
        readonly attribute lastEventId;
        readonly attribute source;

        void initMessageEvent(...);
        void initMessageEventNS(...);
};
```

3.1.3 Events

Um auf Ereignisse des WebSockets zu reagieren, kann der Entwickler Callbacks für vier Events definieren [4]:

onopen(event:Event) Der Callback wird aufgerufen, sobald die TCP-Verbindung geöffnet und ein Handshake mit dem Server erfolgreich durchgeführt wurde.

onmessage(event:MessageEvent) Der Callback wird jedes Mal dann aufgerufen, wenn eine neue Nachricht vom WebSocket-Server eingetroffen ist. Das „data"-Attribut des „event"-Parameters (vgl. Quelltext 2) enthält den Inhalt der Nachricht, der Wert steht abhängig vom übermittelten Datentyp im Format String, ArrayBuffer oder Blob zur Verfügung.

onerror() Der Callback wird aufgerufen, wenn ein ungültiges Frame empfangen und die WebSocket-Verbindung daher geschlossen wurde.

onclose(event:CloseEvent) Der Callback wird aufgerufen, wenn die Verbindung getrennt wurde. Der Parameter „event" enthält das Attribut „code", über das der Statuscode für das Schließen der Verbindung abgefragt werden kann.

doi: 10.2313/NET-2012-04-1_8

3.1.4 Attribute

Jeder WebSocket besitzt außerdem folgende readonly-Attribute [4]:

url Die im Constructor übergebene Server-URL.

readyState Der Statuscode der Verbindung als Zahl zwischen 0 und 3. Quelltext 1 zeigt die entsprechenden Konstanten der Statuscodes.

bufferedAmount Anzahl der aktuell zu sendenden Bytes, die noch nicht übertragen wurden.

extensions Alle Erweiterungen des WebSocket-Protokolls, die im WebSocket verwendet werden.

protocol Das eingesetzte Unterprotokoll, falls vorhanden.

3.2 Anwendungsbeispiel

Mit Hilfe der WebSocket-API können Entwickler von Web-Applikationen effiziente bidirektionale Verbindungen mit WebSocket-Servern aufbauen. Der in Abschnitt 1 angesprochene Overhead durch den Einsatz von HTTP-basierten Technologien entfällt. Dies lässt sich am Beispiel des Browserchats veranschaulichen.

Quelltext 3 zeigt den JavaScript-Code eines einfachen Browserchats. Um den Code übersichtlich zu halten wurde die Bibliothek jQuery[3] verwendet. Der Code enthält eine Funktion „addMessage“, um Nachrichten in einem Container des DOMs anzuhängen, einen Listener, der bei Klicks auf einen „send“-Button eine Nachricht versendet und eine Poll-Funktion, die jede Sekunde einen Ajax-Request an einen Server sendet, um - falls vorhanden - neue Nachrichten zu erhalten.

Quelltext 4 zeigt den selben Chat-Client unter Verwendung von WebSockets. Auf den ersten Blick fällt auf, dass der Code etwas kompakter ist als die Polling-Variante. Die Funktion „wsChat.ondata“ kann per Server-Push neue Nachrichten unmittelbar empfangen, sobald sie im Server eintreffen. Dadurch kommen im Gegensatz zu Quelltext 3 keine leeren Nachrichten mehr an und mehrere Nachrichten müssen nicht mehr als Array gebündelt versendet werden. Prüfung und Iteration über „data“ entfallen daher.

Ressourcenverwendung und Performance

Der eigentliche Mehrwert zeigt sich aber bei der Ressourcenverwendung. In Variante 1 muss jede Sekunde ein neuer Request gesendet werden, unabhängig davon, ob tatsächlich neue Daten vorliegen. Jeder Request erzeugt dabei einen HTTP Header-Overhead. Abbildung 3 zeigt den Anfrage- und Antwortheader einer typischen Ajax-Anfrage, die in diesem Beispiel zusammengenommen eine Länge von 871 Bytes aufweisen [8]. Da die Headerlänge von Anzahl und Umfang der übertragenen Metainformationen abhängt, kann sie je nach Anwendungs-Szenario auch deutlich länger oder kürzer ausfallen. Abbildung 4 zeigt, welchen Netzwerktraffic dieser Headeroverhead in Abhängigkeit von der Zahl der gleichzeitig auf einer Webseite aktiven Benutzer unter folgenden Annahmen generiert:

[3] s. http://jquery.com

Quelltext 3 Browserchat via Ajax-Polling

```
var chatCon = $('#chatContainer'),
    chatInput = $('#chatInput');

function addMessage(message) {
    $('<div class="msg">' + message + '</div>')
        .appendTo(chatCon);
}

function pollForMessages() {
    $.get(
        'http://server.de/chat',
        {},
        function(data) {
            if($.isArray(data))
                return;
            for(var n = 0; n < data.length; n++)
                addMessage(data[n]);
        }
    );
}

setInterval(pollForMessages, 1000);

$('#send').on('click', function() {
    addMessage(chatInput.val());
    $.post(
        'http://server.de/chat',
        {input: chatInput.val()}
    );
    chatInput.val('');
});
```

Quelltext 4 Browserchat via WebSockets

```
var chatCon = $('#chatContainer'),
    chatInput = $('#chatInput'),
    wsChat = new WebSocket('ws://server.de:80/chat');

function addMessage(message) {
    $('<div class="msg">' + message + '</div>')
        .appendTo(chatCon);
}

wsChat.ondata = function(event) {
    addMessage(event.data);
};

$('#send').on('click', function() {
    addMessage(chatInput.val());
    wsChat.send(chatInput.val());
    chatInput.val('');
});
```


doi: 10.2313/NET-2012-04-1_8

- Polling jede Sekunde
- Neue Nachricht alle 2 Sekunden
- Nachrichtenlänge 50 Bytes (entspricht 4 Byte Frame-Header)
- Ein Frame pro Nachricht

Fazit: Der Netzwerktraffic kann bei diesen Header-Längen durch den Einsatz von WebSockets von 83MB/s auf ca. 195 KB/s verringert werden. Die Einsparung von 99,77% der benötigten Netzwerkkapazitäten resultiert aus dem geringen Header-Overhead des WebSocket-Protokolls von nur 4 Byte. Durch das Polling entsteht außerdem eine Verzögerung, bis die Nachrichten beim Chatpartner angezeigt werden, der selbst bei Vernachlässigung der Übertragungszeiten bis zu eine Sekunde dauern kann. Die Verzögerung könnte zwar verkürzt werden, indem das Abfrageintervall verringert wird, allerdings würde dies den Ressourcenaufwand noch weiter erhöhen.

4. WEBSOCKETS „IN THE WILD“

Das WebSocket-Protokoll ist zum aktuellen Zeitpunkt noch sehr jung und wurde von der IETF im vergangenen Jahr mehrfach überarbeitet. Erst am 11.12.2011 wurde das Protokoll als IETF-Standard veröffentlicht. Dementsprechend gering fällt zum aktuellen Zeitpunkt die Unterstützung der Webbrowser aus. Tabelle 1 zeigt die geplante oder realisierte Implementierung entscheidender Protokoll-Entwicklungsstufen in gängigen Webbrowsern.

Protokoll	hixie-76 (Mai 2010)	hybi-10 (Jul. 2011)	RFC 6455 (Dez. 2011)
Chrome	6	14	16
Safari	5.0.1	-	-
Firefox	4.0 (deaktiviert)	6	11
Opera	11.0 (deaktiviert)	-	-
IE	-	IE 10 developer preview	-

Tabelle 1: WebSocket Browserunterstützung [2]

In diesem Abschnitt wird gezeigt, wie WebSockets dennoch schon heute browserübergreifend einen Mehrwert für Webapplikationen bringen können.

4.1 Alternative Technologien

Wie bereits dargelegt wurde, ermöglichen WebSockets gegenüber HTTP eine bidirektionale Kommunikation zwischen Webbrowser und Webserver. In der Praxis haben Webentwickler aber bereits verschiedene technische Alternativen auf der Basis von HTTP eingesetzt, die eine ähnliche Kommunikation ermöglichen.

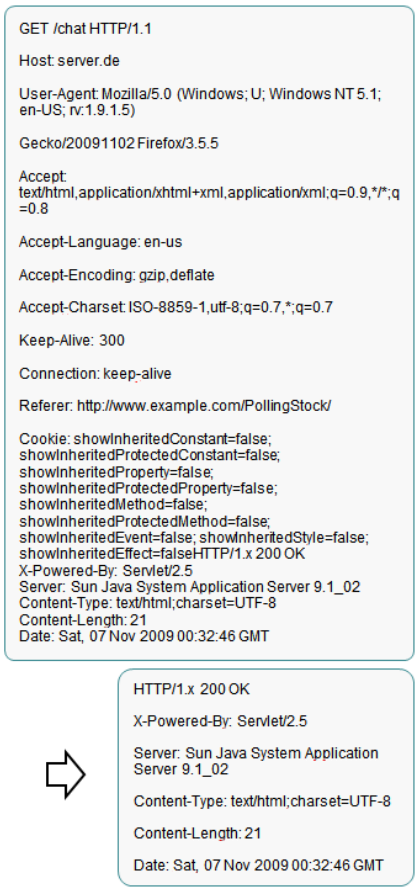


Abbildung 3: Beispiel HTTP-Header [8]

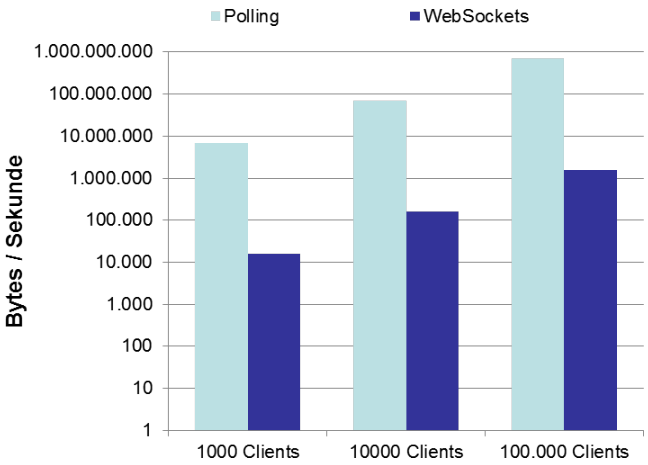


Abbildung 4: Netzwerktraffic Polling / WebSockets (in Bit/s) bei einem Abfrageintervall von 1 Sekunde; logarithmische Skalierung (Basis 10); nach eigener Berechnung

doi: 10.2313/NET-2012-04-1_8

4.1.1 Comet

Ein Ansatz, einen Server-Push ohne WebSockets zu realisieren, ist der Einsatz des Web-Application Models „Comet“[4]. Dieses umfasst verschiedene technische Methoden, HTTP-Requests an einen Server zu schicken, ohne dass dieser die Anfrage sofort beantwortet. Stattdessen wird die Verbindung so lange aufrechterhalten, bis dem Server neue, für die Anfrage relevante Daten zur Verfügung stehen, die dann über den bereits geöffneten Request an den Client geschickt werden können [9]:

Streaming

Eine dieser Varianten ist „Streaming“. Hierzu wird ein HTTP-Request z.B. als Ziel-URL eines iFrames an den Server gesendet. Dieser beantwortet die Anfrage aber nie vollständig, sondern sendet neue Daten immer scheibchenweise, sobald diese verfügbar sind. In einem iFrame lässt sich dies z.B. dadurch realisieren, dass die einzelnen Datenblöcke in JavaScript-Code eingefasst und gesendet werden, der dann direkt im Browser ausgeführt wird. Die Daten können so unmittelbar von der Anwendung verwertet werden.

Longpolling

Eine weitere Technik, um einen Server-Push zu realisieren, ist Longpolling. Wieder wird ein HTTP-Request z.B. via Ajax oder iFrame gestartet, der nicht direkt beantwortet wird. Stehen dem Server neue Daten zur Verfügung, beantwortet er den offenen Request und beendet die Verbindung. Der Client verarbeitet die Daten und schickt sofort eine neue Anfrage, die dann wieder bis zur nächsten Antwort offen gehalten wird. [9]

4.1.2 Sockets über Flash

Eine weitere Möglichkeit um mit WebSockets bereits heute eine größere Reichweite zu erzielen, ist der Einsatz von WebSockets über den Adobe Flash Player. Dieser erlaubt seinerseits bereits seit Version 10 das Öffnen von Socket-Verbindungen zu einem Server. Dies erfordert allerdings eine individuelle Konfiguration des Servers, da z.B. ein zusätzlicher Port in der Firewall geöffnet werden muss. Technische Einschränkungen bestehen außerdem, wenn Proxys eingesetzt werden und der Flash-Player die Proxy-Einstellungen des eingesetzten Browsers nicht auslesen kann. [6]

4.2 Technologie-Fallback via Socket.IO

Wie in diesem Abschnitt gezeigt, existieren verschiedene Techniken, um bei Abwesenheit einer nativen WebSocket-Implementierung des Browsers eine bidirektionale Verbindung zu einem Server aufzubauen oder zumindest zu simulieren. Diese haben jedoch kein einheitliches Interface und können teilweise nur unter bestimmten Voraussetzungen, wie dem Vorhandensein des Flash Players, eingesetzt werden.

Aus diesem Grund wurde die JavaScript-Library Socket.IO[5] entwickelt, die ein einheitliches Interface für den Einsatz von Server-Push Funktionalität anbietet. Je nach Verfügbarkeit verschiedener Technologien des eingesetzten Webbrowsers werden native WebSockets, Flash-Sockets oder Comet-Technologien eingesetzt, um eine möglichst große Abdeckung unter den potentiellen Besuchern einer Web-Applikation zu erreichen.

[4]s. http://cometdaily.com/

[5]s. http://socket.io

5. ZUSAMMENFASSUNG UND AUSBLICK

Mit dem zunehmenden Wandel von statischen Hypertexten zu dynamischen Web-Applikationen erzeugt das HTTP-Protokoll einen Overhead an benötigten Netzwerkressourcen und unnötige zeitliche Verzögerungen bis zur Darstellung aktualisierter Webseiten-Inhalte.

Auf Basis des WebSocket-Protokolls steht Webentwicklern eine schlanke und gleichzeitig mächtige API zur Verfügung. Mit dieser kann der Overhead an verwendeten Ressourcen für Echtzeit-Applikationen minimiert und das Zeitverhalten gegenüber Technologien wie Polling verbessert werden. Durch die Konzeption des Protokolls als Upgrade des bestehenden HTTP-Protokolls können WebSocket-Verbindungen ohne großen technischen Mehraufwand in bestehende Internet-Infrastrukturen integriert werden.

Die Schwierigkeiten bei der Programmierung auf Basis des WebSocket-Protokolls liegen derzeit jedoch in der geringen Browserkompatibilität. Da der WebSocket-Internetstandard zum Zeitpunkt der Arbeit noch sehr neu ist und Webseiten meist Kompatibilitäten mit allen wichtigen Webbrowsern und deren verbreiteten Versionen erfordern, wird es noch einige Zeit dauern, bis native WebSockets für ein breites Publikum eingesetzt werden können. Dennoch lassen sich WebSockets schon heute für Web-Applikationen verwenden, indem die Software je nach Browser des Anwenders alternative Technologien wie Longpolling oder Flash-Sockets nutzt. Bibliotheken wie Socket.IO bieten dem Entwickler hierzu ein einheitliches Interface. Auf dieser Basis entwickelte Anwendungen können so schon heute ein verbessertes Zeitverhalten erreichen. Durch zunehmende Browserunterstützung profitieren so entwickelte Applikation mittelfristig automatisch von der Ressourceneffizienz nativer WebSockets.

6. LITERATUR

[1] HTML 5 - A vocabulary and associated APIs for HTML and XHTML. http://www.w3.org/TR/2008/WD-html5-20080610/comms.html. abgerufen am: 10.02.2012.

[2] WebSocket. http://en.wikipedia.org/wiki/WebSocket. abgerufen am: 10.02.2012.

[3] I. Fette and A. Melnikov. The WebSocket protocol. 2011.

[4] I. Hickson. The websocket api. 2011.

[5] L. Huang, E. Chen, A. Barth, E. Rescorla, and C. Jackson. Talking to yourself for fun and profit. *Proceedings of W2SP*, 2011.

[6] H. Ichikawa. HTML5 Web Socket implementation powered by Flash. http://github.com/gimite/web-socket-js, 2011. abgerufen am: 10.02.2012.

[7] J. Ihlenfeld. Websockets werden mit dem RFC 6455 zum Internetstandard. (Bild: IETF). http://www.golem.de/1112/88360.html, 12 2011. abgerufen am: 10.02.2012.

[8] P. Lubbers. Harnessing the Power of HTML5

WebSocket to create scalable Real-Time Applications.
[9] S. Platte. AJAX und Comet. 2010.

doi: 10.2313/NET-2012-04-1_8

ISBN 3-937201-26-2
DOI 10.2313/NET-2012-04-1

ISSN 1868-2634 (print)
ISSN 1868-2642 (electronic)