2013 27th International Conference on Advanced Information Networking and Applications Workshops

SWIFT: A Secure Web Domain Filter in Hardware

Vlado Altmann, Jens Rohrbeck, Jan Skodzik,
Peter Danielis, Dirk Timmermann
University of Rostock
Institute of Applied Microelectronics and Computer Engineering
18051 Rostock, Germany
Tel./Fax: +49 (381) 498-7257 / -1187251
Email: vlado.altmann@uni-rostock.de
Web: http://www.imd.uni-rostock.de/networking

Maik Roennau, Matthias Ninnemann
Nokia Siemens Networks GmbH & Co. KG
Broadband Access Division
17489 Greifswald, Germany
Tel: +49 (89) 5159-16117
Email: {maik.roennau;matthias.ninnemann}@nsn.com

***Abstract*—The risk of becoming a victim of spam and phishing attacks increases in today's Internet. Various Web sites exhibit violent or illegal content. Unfortunately, many users are not able to protect themselves and their networks. Already now, some countries deploy systems to filter Web content. However, the existing solutions show high latency or overblocking. Thus, a novel Web filtering concept to protect users at the Internet service provider level is presented. The proposed system is able to detect and block illegal and threatening Web sites. The suggested scalable hardware-based approach can examine Internet domains in wire speed without overblocking. The Web filter serves as security measure for all connected users, especially for users with limited IT expert knowledge. The system is fully transparent for all network devices. Setup and maintenance can be made only by the Internet service provider administrator. Consequently, the suggested security system itself is safe from attacks from users and from the network side.**

I. Introduction

Providing network security is one of the most important tasks in today's Internet. There are plenty of threats such as viruses, malware, and phishing that are able to harm Internet users. Moreover, violent or illegal content such as child pornography can be found on the Internet.

Users use antivirus programs and firewalls to protect their computers and networks against malicious software. However, these kinds of programs do not give protection against malicious Web content. As preventive measure against this type of content, Web filters can be used. Normally, these security measures are installed on computers of users. But installing security measures at the users' side has two serious drawbacks. Firstly, threat detection is done on the target machine, which is often already infected with malicious software. Secondly, the users must install, upgrade, and maintain these security measures without professional support. Especially, a Web filter is often not installed at all and requires additional maintenance. Moreover, the majority of Internet users is missing the necessary expertise to configure their security software so that it provides optimal protection. Therefore, it is mandatory to support users in issues of Internet security by means of a Web filter.

A trustworthy place for the placement of a Web filter is the ingress of the network – the access network. Each user, referred to as subscriber by Internet Service Providers (ISPs), is connected to the Internet through the access network. The access network itself consists of access nodes (ANs) [1]. As ANs are transparent for subscribers and the core network, these components are safe from, e.g., Denial of Service (DoS) attacks. In order to provide protection for subscribers, an AN is the optimal place to establish additional security service such as Web filter. With this additional feature, the subscriber does not need to care about security measures himself.

However, although an ISP can take up new security measures in its portfolio, various challenges have to be addressed:

- On an AN, very high traffic rates (e.g., 1 Gbit/s or higher) have to be processed.
- The Web surfing experience of subscribers must not be constrained.
- Web sites, which are not malicious, must not be falsely blocked by the Web filter.

To perform Web filtering under the conditions described, very powerful packet classification and packet processing are required. Due to these requirements, pure software solutions are not applicable. Therefore, SWIFT — a **S**ecure **W**eb Domai**n** Fil**t**er in Hardware — has been developed. A prototype was built on a XILINX evaluation board with a FX70T Field Programmable Gate Array (FPGA). In the suggested solution, content-addressable memory (CAM) is not used as already for 160 domain names, ca. 88% of available block random access memory (BRAM) resources or 23% of slice registers would be consumed. However, without using CAM, the system is able to control traffic in wire speed.

The suggested approach was developed as part of the Secure Access Node (SecAN) project presented in [2] as "work in progress". SecAN is devoted to protection of ordinary users in access networks. It deals with firewalls, intrusion detection systems and Web filters. Thereby, Web filtering functionality moves from the subscriber to the ISP. This paper elaborates on the Web filter approach highlighting its benefits concerning speed, resource consumption, scalability, and distinct pattern matching. Briefly summarized, the main contributions of this paper are the following:

- A novel scalable hardware approach of a Web filter placed onto an AN is presented.

978-0-7695-4952-1/13 $26.00 © 2013 IEEE
DOI 10.1109/WAINA.2013.15

- The suggested architecture does not produce false positives. Thus, only blacklisted domains are blocked.
- The system is able to control traffic with more than 1 Gbit/s in wire speed on the target platform without packet loss.

This work is structured as follows. In Section II, related work is presented. Section III describes the functionality of the hardware Web filter. In Section IV, the implementation details of SWIFT are addressed. Before the paper concludes in Section VI, achieved results are introduced in Section V.

II. Related Work

Today, Web filtering systems are used in Great Britain, USA, China, and other countries [3]–[8]. These projects are normally not published in detail due to government and security restrictions. Contrary, in the authors' opinion this paper represents a significant contribution raising the comprehensibility of hardware-based security mechanisms. The British system "Cleanfeed" has a two stage structure [3]. In the first stage, the system filters IP addresses. If the IP address matches a filter entry a request is sent in the second stage to the external data base to verify the domain. The data base is managed by the Internet Watch Foundation, which collects reports about criminal online content. "Cleanfeed" grants an efficient domain filtering. However, it suffers from high latency due to its structure, which constrains the Web surfing experience of users. SWIFT avoids this drawback by solely utilizing local resources ensuring high processing speed. The US Web filtering system achieves better latency than "Cleanfeed". However, overblocking was substantiated, i.e., the Web sites were blocked although they were not blacklisted [4]. SWIFT does not produce false positives as each domain is exactly verified in the blacklist. Thereby, overblocking is avoided. The China Internet filtering system inspects Web traffic for specified keywords [5]. If the keyword is found the Web filter resets the connection by setting the TCP reset flag. The frame that contains the keyword is still forwarded to the recipient. If the endpoints ignore the reset flag the connection persists. Thus, the content can be transported to the requester. As opposed to this approach, SWIFT drops the packet with the request to malicious Web content and though the communication is interrupted.

There are also commercial Web filtering solutions available on the market such as Barracuda Networks [9], Blue Coat [10], Websense [11], and others. These solutions are mainly all-in-one products targeting intranet and enterprise networks. They follow a gateway concept and are implemented as software-hardware co-design. Although interfaces, which are capable of 1 Gbits/s, are available, the maximal reachable throughput is limited to 300 Mbit/s [9]. Solutions for the core network such as Huawei Service Inspection Gateway (SIG) adopt the application-specific integrated circuit (ASIC) and network processor architecture for processing packets and the ASIC, FPGA, and multi-core architecture for processing services. SIG provides varieties of interfaces up to 10 Gbit/s [12]. Thereby, up to 80 million uniform resource locators (URLs) per day can be processed [13]. This corresponds to the worst case throughput of Web traffic of some Mbit/s. An FPGA-based Web filter was proposed in [14]. The authors utilize ternary content-addressable memory (TCAM) and an FPGA for the packet preprocessing and a general purpose computer for the final processing. The practical drawbacks of this solution are an expensive TCAM and an additionally required hardware such as computer. FPGA resource consumption and frequency are not mentioned. In contrast to the mentioned solutions, SWIFT reaches full 1 Gbit/s throughput even in the worst case. Moreover, it does not require any additional devices for packet processing.

III. The Hardware Web Filter

Each subscriber is connected to the Internet by the access network. Access networks comprise access nodes such as Digital Subscriber Line Access Multiplexers. Because the Web filter is located on an AN, a bandwidth of at least 1 Gbit/s must be achieved. Therefore, the Web filter was designed and developed as hardware solution since software cannot achieve the necessary throughput.

A. Web Filtering

In order to protect ordinary Internet users, e.g., against malicious Web content, there are three general possibilities: Either IP addresses, URLs, or domain names can be controlled. The filtering of IP addresses can lead to crucial overblocking since different Web sites can share the same Web server. Moreover, a Web site can move to a new Web server while it remains accessible over the old domain name. For these reasons, only URL and domain name filtering are concerned. The domain name is part of each URL. A domain name points to the server, whereas URL points to the specific resource on the server. URL filtering can lead to underblocking, i.e., the path on the server to a requested resource (e.g., an image on the Web site) can be newly created on every access. Therefore, URL filtering is not efficient against malicious content. Domain names represent a fixed part of the URL and thus can not be easily changed. From the hardware view, URLs are disadvantageous as their maximum length is not defined [15]. Moreover, URL characters can be encoded in different ways. Since all possibilities must be captured in hardware, the hardware complexity increases enormously. In contrast, the tree-like structure of domain names is hardware-friendly [16]. A domain name has a defined structure and may be up to 255 characters long. The characters are ASCII encoded. Furthermore, only letters (a-z), digits (0-9), period (”.”), and hyphen (”-”) are allowed. Since the focus of this paper is to protect unexperienced users against malicious content using a high speed hardware architecture, it is advantageous to check domain names rather than URLs.

Domain filtering can be done on Domain Name System (DNS) basis or Hypertext Transfer Protocol (HTTP) basis. Table I shows the advantages and disadvantages of the two possibilities with respect to domain filtering. HTTP-based filtering grants immediate effect, which is an essential issue

SELECTION CRITERION	DNS	HTTP
Exclusive monitoring of HTTP traffic	No, other protocols will be blocked as well (e.g., FTP).	Yes, only HTTP traffic will be blocked.
Immediate effect	No, due to the Time-to-Live value of the DNS	Yes, there is no time dependence in HTTP requests.
Can be avoided by HTTP proxies	Yes, since HTTP proxy acts as broker.	No, the domain name is included in HTTP request message to the proxy.
Simple bypass of Web filter possible	Yes, if the searched IP is known (e.g., in local hosts file) no DNS request will be send.	No, to get the Web content, a HTTP request is required.

TABLE I
WEB FILTER SELECTION CRITERIA — DNS VS. HTTP

in order to block a malicious Web content. Moreover, HTTP requests to proxies are checked as well. HTTP redirections as used by URL shorteners are captured too, since the final request is still sent to the target domain. As the goal is to monitor Web traffic, other protocols should not be blocked. Using HTTP monitoring, the Web filter cannot be simply bypassed by adding the IP address of the Web server into the local hosts file. Based on the results, which highlight the benefits of HTTP filtering, it is advantageous to check domain names in HTTP requests.

Domain names can be found in two places within the TCP payload. The first one is the HTTP request line. For example:
GET http://www.w3.org/WWW/TheProject.html HTTP/1.1
This kind of request contains the URL in absolute form and is usually used to get a Web site from a HTTP proxy. The absolute URL can be detected by the ”http://” pattern following the ”GET” pattern. The other possible position of a domain is within the Host header. In this case, the request line only contains a relative part of the URL. For example:
GET /WWW/TheProject.html HTTP/1.1
Host: www.w3.org
A relative URL can be detected by the slash character (”/”) after the ”GET” pattern. This kind of request is used to get the Web site directly from Web server. As end mark of each HTTP header, Carriage Return (CR) and Line Feed (LF) are used.

However, the Web filter can still be bypassed using tunneling or encryption. These methods are generally problematic for all deep packet inspection systems making pattern matching impossible. Another way to bypass the Web filter is using Really Simple Syndication (RSS) feed from an online reader [17]. However, this method has several limitations. Firstly, the blocked Web site must provide RSS feed, which URL is explicitly known by the user. Secondly, only text from the RSS feed will be shown. All other resources, e.g, images must be requested directly from the blocked host and thus are blocked. As the main purpose of this work is the protection of ordinary users without expert knowledge from malicious Web content, the mentioned bypass methods are not relevant.

B. General Functionality

The system is placed in upstream direction, i.e., checks traffic from Internet users towards the core network. Before the system can process traffic, it must be configured. During this process, the configuration data (i.e., domain blacklist) are stored in the Web filter memory. After configuration, the frame processing starts. A HTTP request frame is detected by the ”GET” pattern in TCP payload. Non-HTTP frames are passed through unchanged. Otherwise, the domain name is searched in the TCP payload. Unlike IP or TCP, HTTP is a text-based protocol [18]. Therefore, a hardware text parser is required. The SWIFT text parser can search and extract the domain on the fly. Thereby, the domain name is stored in a cache memory. For caching purposes, BRAM of the FPGA is used. Afterwards, the actual domain name is searched in the blacklist. If a match is found the frame with the HTTP GET request is dropped. Otherwise, it is passed through.

IV. IMPLEMENTATION

The goal of this work is to implement a Web filter considering following requirements:

- Distinct domain matching: No false positives should be possible
- High speed: At least 1 Gbit/s should be achieved
- High scalability: The system should scale with growing blacklist
- Low resource consumption: Efficient usage of FPGA resources should be achieved
- Network transparency: The system should be transparent to all network participants

To meet these requirements, a fast, scalable, and resource-efficient searching approach is necessary. For this purpose, a two level search architecture is proposed. Thereby, the 1st level search represents a pre-filter and the 2nd level search is the final filter.

A. 1st level search architecture

For the 1st level searching, hashing is used. While the Web filter processes and caches the domain name, the hash value is calculated in parallel. The hash table is stored in a cache memory (BRAM) and thus access to it is very fast (one cycle). As hash function, parallel CRC64 was chosen as it provides an optimal resource consumption in term of speed/collision ratio. The size of the hash table corresponds to the blacklist length. The hash values are stored in an ascending order beginning from address 0x00. For storing hash values, a heap structure is used. The search starts in the middle of the occupied range, i.e., in the root of the heap (see Figure 1). The middle value splits the table in two subtables. Thereby, the middle value of each subtable is the root of the corresponding subtree. The left subtree contains all hash values less than the root and the right subtree those greater than the root. The splitting continues until no further subtrees can be built. In order to find a match, the calculated hash value is compared with the root. If the current root value is bigger the left subtree will be taken. Otherwise, the search continues in the right subtree. Subsequently, the root

of the subtree is compared with the calculated hash value. If the leaf of the tree is reached and no match was found, the search is not successful.

Address	Hash Value
0x06	7
0x05	6
0x04	5
0x03	4
0x02	3
0x01	2
0x00	1

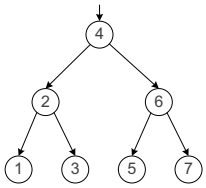

Fig. 1. Heap structure of the 1st level search architecture

A heap search has a logarithmic complexity. Thus, the suggested approach scales with growing blacklists. If the amount of domains doubles only one additional cycle is required to fully traverse the hash table.

If the 1st level search is successful the corresponding memory address of the found hash value is returned. Since the 1st level search performs pre-filtering, a 2nd level search is required in order to verify the domain in the blacklist. In case the search is not successful, the frame will be forwarded towards the system output.

B. 2nd level search architecture

The 2nd level search architecture comprises plain text domains and additional meta information. The architecture shows a buckets structure. Each bucket contains a domain list with information on domains producing the same hash value. An example is depicted in Figure 2.

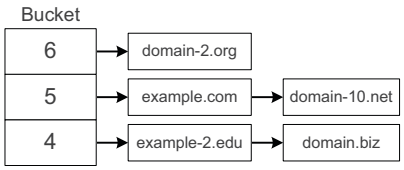

Fig. 2. Buckets structure of the 2nd level search architecture

This structure is stored in DDR2 SDRAM. This is an external memory and thus searching in DDR2 SDRAM is slow (ca. 20 waiting cycles for first data are required). In order to accelerate domain matching, additional meta information such as domain length, collision flag, and next address is stored as well. An exemplary memory configuration is shown in Figure 3.

Phys. Addr.	Length	Collision	Next Addr.	Domain
0x08	13	false	0x00	domain-10.net
0x07	10	false	0x00	domain.biz
0x06	12	false	0x00	domain-2.org
0x05	11	true	0x08	example.com
0x04	13	true	0x07	example-2.edu

Fig. 3. Exemplary DDR2 SDRAM configuration

If the 1st level search is successful the DDR2 SDRAM address of the bucket (start search address) is calculated by the extracted address of the hash value. For this purpose, following formula is used:

$$A = A_{hash} \cdot L_{entry} + A_{start}$$

where A_{hash} is a hash address extracted in the 1st level search, L_{entry} is the constant amount of memory cells required to store one entry, and A_{start} is the starting address of the blacklist. On the target platform, 34 memory cells are required to store one entry. If data is requested, the memory provides it from two cells per cycle (due to double data rate) after the already mentioned waiting cycles. Though, the reading can be interrupted if the memory has to be refreshed. Therefore, meta information is read within the first cycle.

The length of the domain is compared first in order to avoid unnecessary reading. The collision flag indicates the presence of other domains in the bucket and the next address field specifies their location in the memory. If the length of the domain does not match and the collision flag is set the next address will be immediately requested from the memory. At this address, the next bucket element is stored. If the length matches the full domain must be compared. As a consequence of the domain match, the current frame is dropped since a HTTP GET request was sent to a blacklisted Web site. The search is not successful only if the end list element of the bucket was reached and no match was found.

For instance, if the requested domain is *domain.biz* (see Figure 3), then the calculated bucket address is 0x04. The length comparison yields no success but the collision flag indicates the presence of other domains in the bucket. The next address field points to the next element in the domain list. Thus, the data from the memory address 0x07 will be requested. As the length matches, the full domain will be compared as well. The domain match implies in that case that the current frame will be dropped. The described data flow is visualized in Figure 4.

In summary, the DDR2 SDRAM is logically divided into two parts: a head memory and collision memory. The head memory contains the head elements of the buckets and can be directly accessed after the 1st level search. The collision memory contains the domain list (excluding the head element). Due to the suggested architecture, a bucket cannot be empty. Since collision memory is directly attached to the head memory, there are no memory voids. This results in the best possible memory utilization.

C. SWIFT Configuration

To provide a high degree of flexibility, the blacklist of the Web filter is configurable. Since the configuration process is not time critical, this task is fulfilled by software that run on an ordinary computer. Configuration data is sent to SWIFT over a dedicated secured channel (e.g., dedicated Ethernet line) and is flashed to both memories. The old configuration is overwritten in this process. Hence, if one domain address was added to or deleted from the blacklist SWIFT must be reflashed. Blacklist

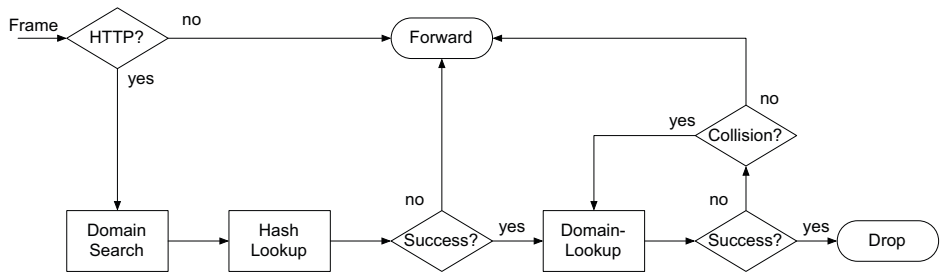

Fig. 4. Web filter data flow

generation and update frequency is on the responsibility of the network administrator or ISP.

V. Results

In this section the achieved results are discussed considering stated requirement for SWIFT (see Section IV).

Distinct domain matching: For the SWIFT prototype, the blacklist length is limited to 4096 domains. Since cache memory is used to store hash values, maximal 12 cycles are necessary to fully traverse a tree. In order to test SWIFT, a data bank with real world domains provided by the domain name registrar VeriSign was used [19]. 23 million domains were hashed with the CRC64 hash function and thereby, 159 collisions were detected with a maximum of two domains per collision. The collision rate is $6.9{\cdot}10^{-6}$. As a result, one bucket would normally have only one domain and thus, only one DDR2 SDRAM access is required in the 2nd level searching. According to that, the possibility to get a false positive in the 1st level searching is below 1‰. In the 2nd level searching, the domain name is exactly verified in the blacklist and thus, distinct domain matching is achieved.

High speed: To avoid discarding any data frame due to the internal delay during frame processing as well as to optimize the workload of the hardware, the internal throughput is set to 32 bit/cycle. The prototype achieves 148.5 MHz corresponding to 4.75 Gbit/s internal throughput. The processing of the Web filter's blacklist induces indispensable computation and waiting cycles. In the worst case, 12 cycles cache computation, 37 waiting cycles for DDR2 SDRAM access including refresh cycles, and 33 cycles for DDR2 SDRAM processing if collision resolution is necessary. The overall domain lookup would take 82 cycles in the worst case.

For performance evaluation, the worst case is considered. Short HTTP GET frames with short domains would evoke blacklist lookups more often and thus, lead to a higher delay. The shortest domain name (e.g., *"g.cn"*) can be found after 81 processed bytes. This includes 14 bytes ethernet header, 20 bytes IP header, 20 bytes TCP header, and first 27 bytes of HTTP headers (HTTP request line and host header). This data volume can be internally processed within 21 cycles. Thus, the overall packet processing time would take 103 cycles (bytes processing plus domain lookup). In order to avoid packet loss, the overall internal packet processing time must be less than or equal to the time for receiving this packet. Consequently, the following equation must be satisfied:

$$T_{proc} \leq L_e \cdot t_e, \tag{1}$$

where T_{proc} is internal processing time of the frame, L_e is frame length, and t_e is the time for receiving 1 byte. Incoming frames are firstly stored in a 2 KB buffer. The length of the considered worst case HTTP GET frame on the wire is 106 bytes. This includes 7 bytes preamble, 1 byte start frame delimiter, 14 bytes ethernet header, 20 bytes IP header, 20 bytes TCP header, 28 bytes HTTP header, 4 bytes frame check sequence, and 12 bytes interframe gap. The time to receive 1 byte at 1 Gbit/s amounts to 8 ns. Thus, T_{proc} must be less than or equal to 848 ns (according to Equation 1). Since internal processing takes 103 cycles, the minimum required frequency corresponds to 121.5 MHz. SWIFT reaches 148.5 MHz. Thus, it can process traffic at full 1 Gbit/s without packet loss.

High scalability and low resource consumption: Resource consumption for the test platform (XILINX FX70T) is depicted in Table II. The hardware synthesis was performed with XILINX XST Synthesizer. Growing blacklists result in increasing BRAM and DDR2 SDRAM consumption as shown in Figure 5, which cannot be avoided. However, the slice consumption remains constantly low pointing out the efficient hardware resource utilization of SWIFT. Due to the tree structure of the 1st level searching, doubling the blacklist size induces only one additional search cycle. That is, the search speed only slightly decreases when storing significantly more blacklist entries.

Resource	Available	Consumption	
		Absolute	[%]
Slice Register	44800	5003	11%
Slice LUT	44800	5957	13%
BRAM	148 (x36Kbit)	28 (x36Kbit)	19%
DDR2 SDRAM	512 MB	1,1 MB	0,2%

TABLE II
SWIFT RESOURCE CONSUMPTION

Network transparency and security: The suggested solution is absolutely transparent for all network participants. All frames except for blacklisted HTTP GET requests are forwarded unchanged. The system configuration is done by a network administrator over a dedicated channel. SWIFT

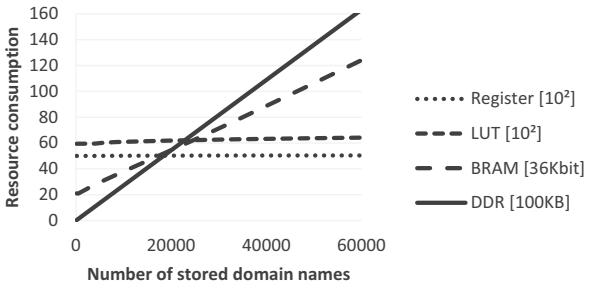

Fig. 5. Resource consumption as a function of the stored number of domain names

represents a pure hardware solution without running a software or operating system. Therefore, it shows full deterministic behavior. Moreover, it can process traffic with full wire speed. Consequently, the system is hardly vulnerable against attacks such as DoS or attacks targeting some flaws of the operating system.

VI. Conclusion and Outlook

In this paper, the working prototype of the hardware Web filter was presented. As hardware solution, it offers more advantages in terms of security and robustness than a software solution. SWIFT can control HTTP traffic in wire speed without packet loss. On the test platform, 1 Gbit/s throughput is achieved. The throughput is only limited by the FPGA type and can be even multiplied by using ASIC. As a high speed Web filter, it can be deployed in the access networks of ISPs. Hereby, the suggested solution can protect users without IT expert knowledge from illegal, aggressive, or threatening Web contents such as child pornography or phishing. SWIFT does not produce false positives. Thus, only blacklisted domains will be blocked. The configuration of the blacklist can be done only by the network administrator. Since SWIFT is fully transparent for all network participants, it is safe from attacks. Prospectively, the functionality of the Web filter can be extended to URL filtering allowing to block specific resource, e.g, an image on the server.

Acknowledgment

The authors would like to thank the former Broadband Access Division of Nokia Siemens Networks GmbH & Co. KG in Greifswald, Germany for their inspiration and support in this project. This work is partly granted by Nokia Siemens Networks.

References

[1] S. Ooghe, N. Voigt, M. Platnic *et al.*, "Framework and Requirements for an Access Node Control Mechanism in Broadband Multi-Service Networks," Internet Requests for Comments, RFC Editor, RFC 5851, May 2010. [Online]. Available: http://www.rfc-editor.org/rfc/rfc5851.txt

[2] J. Rohrbeck, V. Altmann, S. Pfeiffer, and D. Timmermann, "Secure Access Node: an FPGA-based Security Architecture for Access Networks," *ICIW*, 2011.

[3] R. Clayton, "Anonymity and traceability in cyberspace," *ACM SIGACT News*, vol. 36, no. 653, pp. 115–148, Nov. 2005.

[4] US District Court for the Eastern District of Pennsylvania, "CDT, ACLU, Plantagenet Inc. v Pappert," *337 F.Supp. 2d 606*, September 2004.

[5] R. Clayton, S. J. Murdoch, and R. N. M. Watson, "Ignoring the great firewall of china," *6th Workshop on Privacy Enhancing Technologies*, no. 16, June 2006.

[6] Telenor Norge, "Telenor and kripos introduce internet child pornography filter," *Telenor Press Release*, September 2004. [Online]. Available: http://presse.telenor.no/PR/200409/961319_5.html

[7] King Abdulaziz City for Science and Technology, "Local content filtering procedure," *Internet Services Unit*, 2004. [Online]. Available: http://www.isu.net.sa/saudi-internet/contenet-filtring/filtring-mechanism.htm

[8] The OpenNet Initiative, "Internet filtering in burma in 2005: A country study," 2005. [Online]. Available: http://opennet.net/sites/opennet.net/files/ONI_Burma_Country_Study.pdf

[9] Barracuda Networks, "Barracuda web filter," 2012. [Online]. Available: http://www.barracudanetworks.com/ns/downloads/Datasheets/Barracuda_Web_Filter_DS_US.pdf

[10] Blue Coat ProxySG, 2012. [Online]. Available: http://www.bluecoat.com/products/proxysg

[11] Websense Security Labs, "Web security gateway solutions," 2012. [Online]. Available: http://www.e92plus.com/Libraries/Websense_Documents/datasheet-web-security-gateway-solutions-en.sflb.ashx?download=true

[12] HUAWEI, "Service inspection gateway," 2012. [Online]. Available: http://www.huawei.com/products/datacomm/catalog.do?id=3685

[13] HUAWEI, "Greennet solution brochure," 2012. [Online]. Available: http://www.huawei.com/products/datacomm/catalog.do?id=3596

[14] Y. Hong and D. Kim, "Implementation of fpga card in content filtering solutions for securing computer networks," *ICIC International*, vol. 1, no. 1, pp. 71–76, September 2010.

[15] T. Berners-Lee, "Uniform Resource Locators (URL)," Internet Requests for Comments, RFC Editor, RFC 1738, December 1994. [Online]. Available: http://www.rfc-editor.org/rfc/rfc1738.txt

[16] P. Mockapetris, "Domain Names - Implementation and Specification," Internet Requests for Comments, RFC Editor, RFC 1035, November 1987. [Online]. Available: http://www.rfc-editor.org/rfc/rfc1035.txt

[17] HiR Information Report, "Web filter evasion part 1: RSS and You," April 2008. [Online]. Available: http://www.h-i-r.net/2008/04/web-filter-evasion-part-1-rss-and-you.html

[18] R. Fielding, U. Irvine, J. Gettys *et al.*, "Hypertext Transfer Protocol – HTTP/1.1," Internet Requests for Comments, RFC Editor, RFC 2616, June 1999. [Online]. Available: http://www.rfc-editor.org/rfc/rfc2616.txt

[19] Verisign, Inc., www.verisign.com, 2010.