1. ***J. Computer Network and Information Security,* 2015, 5, 1-8** Published Online April 2015 in MECS (<http://www.mecs-press.org/)> DOI: 10.5815/ijcnis.2015.05.01

Embedded Real-Time HTTP Server

# Radosław Czarnecki

Division of Computer Science, Cracow University of Technology, Poland Email: [czarneck@pk.edu.pl](mailto:czarneck@pk.edu.pl)

# Stanislaw Deniziak

Department of Computer Science, Kielce University of Technology, Poland Email: [s.deniziak@computer.org](mailto:s.deniziak@computer.org)

***Abstract***—This paper presents the architecture of embedded real-time web server. Unlike existing web servers, in our approach, requests are processed not in the

―first in first out‖ order but according to their deadlines and the expected server load. For this purpose the Least Laxity First scheduling method is used. First, requests with imposed hard real-time constraints are served. Then requests enclosed by soft deadlines are processed. Finally, request without time requirements are served in the order they arrived. We also present real-time extensions to the Hypertext Transfer Protocol. We propose headers that enable defining hard and soft deadlines, as well as responses containing time information, that are being sent to the client application. The experimental results showed that in case of real-time applications our server misses significantly fewer requests, due to time out, then existing solutions. The presented server may be very useful for implementing real-time services supported by embedded systems, e.g. in future real-time ―Internet of things‖ applications.

***Index Terms*—**Web server, embedded system, real-time system, HTTP, Internet of things, Sensing as a service.

* 1. INTRODUCTION

More and more embedded systems are connected to Internet. The emerging concepts, like Internet of Things (IoT) [1], Machine to Machine communication (M2M) [2], Sensing as a Sevice (S2aaS) [3], wireless sensor networks [4] etc., stimulate the development of web- enabled devices. Such devices can communicate with web-applications using HTTP protocol [5], a lot of them are controlled by web browsers. For this purpose an embedded web server (EWS) should be built in. EWS usually is a lightweight application implementing only main methods defined by the HTTP protocol. It contains simple web pages, with forms used for the interaction. EWS are used for controlling network printers, wireless routers, network cameras and other devices or sensors. It is expected that in a few years almost each product may be identified and traced on Internet using wireless communication methods. Moreover, a lot of them will supply sensing data to web applications.

Existing web-based sensing applications do not consider time requirements. Although the increasing

number of sensing services will cause that sensors with built in EWS will face real-time conditions, which should be satisfied to provide appropriate and required level of Quality of Service (QoS). Recently it was revealed that future Internet of Things or Service Oriented Architectures should address the real-time aspects. Some web applications have time-critical demand, especially in domains like environmental monitoring, transportation. The IoT will comprise billions of intelligent communicating ―things‖ or Internet Connected Objects (ICOs) that will have sensing, actuating and data processing capabilities. Each ICO will have one or more embedded sensors that will capture potentially enormous amounts of data. To enable processing a large number of requests, such ICO should take into consideration real- time constraints.

Real-time applications expect responses from sensors or external services in predictable time periods. Unfortunately HTTP does not support real-time constraints. When the response to the request did not be received during the expected time it is not possible to determine if it was caused by the EWS overload, network faults, server failure or heavy network traffic. Moreover, in existing HTTP servers it is not possible either to control the order of processing of incoming requests or to predict the time of processing. In real-time environment EWS should first, process requests according to their deadlines, second, if it is not able to process request in the expected time, EWS should send timeout message to a client application. Server that meets above requirements will significantly improve QoS in real-time IoT systems, especially when an ICO supports sensing data for a large number of real-time web applications and when getting these data requires time-consuming computations.

In this work we present the architecture of an embedded real-time HTTP server. To enable imposing the real-time requirements for HTTP requests, we will define the real-time extensions of the HTTP protocol. The server schedules all requests according to their priority, which is based on real-time requirements, and an expected processing time. The main appliance of our server would be embedded systems supporting sensing data in real time. According to our best knowledge there are not similar solutions for embedded systems.

The rest of this paper is organized as follows. In the next section related works are presented. In section 3 we present real-time extensions to the HTTP protocol.

**2** Embedded Real-Time HTTP Server

Section 4 describes the architecture of the real-time HTTP server. The experimental results showing the advantages of applying our server in real-time web applications are given in section 5. Finally, in section 6, we end with some conclusions and show further work to develop in this field.

* 1. RELATED WORK

In [6] the problem of real-time requirements in the Web of Things (WoT) applications was discussed. Authors observed that a lot of WoT systems interact with embedded devices and expect real-time data, thus development of WoT application that satisfy real-time requirements is one of the main challenges. Although some technologies for real-time communication (e.g. RTP/RTSP [7], XMPP [8]) or real-time interaction (e.g. Comet [9]) were developed, but still more developments and standards are required for real-time WoT and IoT systems.

Real-time web applications are often considered in the context of cloud computing. Current work concerning real-time cloud computing (RTCC) mainly concentrates on 2 domains: adopting existing web technologies to this new paradigm and developing software architectures for real-time applications. Recent studies have been performed on the allocation of resources for real-time tasks. Aymerich *et al.* [10] developed an infrastructure for a real-time financial system based on cloud computing technologies. Liu *et al.* [11] showed how to schedule real-time tasks with different utility functions. The real-time tasks are scheduled non-preemptively with the objective to maximize the total utility by using the time utility function (TUF). Tsai *et al.* [12] discuss a real- time database partitioning on cloud infrastructures. Kim *et al.* [13] investigate power-aware provisioning of resources for real-time cloud services. In their work the real-time constraint is specified in a Service Level Agreement (SLA) between customers and cloud providers. SLAs specify the negotiated agreements, including QoS. In such cloud models the service provider is responsible for the allocation resources. Their work examines power management while allocation of resources should meet the SLA. None of the above studies consider a cost-efficient selection, from a set of different types of resources available in clouds, for real- time tasks. Kumar *et al.* [14] developed an algorithm of resource allocation for applications with real-time tasks. They propose an EDF-greedy scheme that considers temporal overlapping to allocate resources efficiently. The methods of cost-aware synthesis of real-time cloud applications for IoT are presented in [15] [16] [17].

There are a lot of implementations of embedded web servers. Appweb [18] is a compact, multi-threaded server that supports in-memory modules for the ESP, Ejscript and PHP frameworks. Fusion Embedded HTTP Server [19] supports only GET and POST methods. It may process multiple concurrent requests and main advantage is very small memory footprint. The Barracuda Web Server [20] is an industrial-strength, small embeddable

web server engine that is optimized for compact, deeply embedded devices. Smews [21] is a prototype of very efficient and very small web server for WoT systems.

There are no known web servers that consider real- time constraints. The Chloe [22] which is called the realtime web server, deals with „real-time web‖, i.e. solutions that enable browsers to receive information as soon as it is published by its authors. The real-time web is fundamentally different from real-time computing since there is no knowing when, or if, a response will be received.

* 1. REAL-TIME HTTP

HTTP is a stateless protocol for communication between a client and a server [Fie99]. An HTTP session consists of a sequence of request-response transactions. The client application sends a request, which may correspond to one of the following methods: GET, HEAD, POST, PUT, DELETE, OPTIONS, TRACE and

CONNECT. As a response the server application sends back a message containing a status line and a requested resource. Not all methods are required to be implemented in the server application. Embedded systems usually use lightweight web servers where the most important is a small memory footprint, minimal CPU utilization and reliability.

Although some extensions for specification of time requirements were proposed [23], the HTTP protocol does not consider real-time requirements. Requests are processed in the FIFO order, regardless of the expected processing time. Thus it is not possible to predict when the client will receive the response, as well as it is not possible to define time constraints associated with the requests. The server latency depends on the number of requests, that must be processed, and on the processing time.

In order to enable specification of real-time constraints, we propose the real-time extensions to the HTTP protocol (RT-HTTP). All methods defined in the HTTP/1.1 are available in the RT-HTTP, moreover, three new headers (Hard Deadline, Soft Deadline and Remaining Time) as well as four new responses (120 Server Timeout, 220 Constraint Satisfied, 420 Wrong Deadline, 520 Deadlines Not Supported) are added.

1. *Header "Hard Deadline"*

Header ―Hard Deadline‖ defines the hard time constraint, i.e. maximal time for processing the request by the server. This header may be specified only in requests. Hard constraint must not be violated, otherwise the server should return the response 120 (Server Timeout). When the request will be processed in time, then the server sends the 220 (Constraint Satisfied) response to the client. The ―Hard Deadline‖ request header is defined in RT-

HTTP as follows:

*Hard-Deadline :=* **Hard Deadline:** SP *time [***ms***]* CRLF where: SP is a whitespace character and CRLF denotes

Embedded Real-Time HTTP Server **3**

the end of line, *time* means a deadline value given in seconds (default) or milliseconds.

Assume that the IoT application traces bus positions in a public transportation system. Each bus is equipped with GPS and EWS supporting the current bus position. To get up-to-date bus position, information should be provided timely. Below, a sample POST request containing hard deadline, is shown:

*POST /task/GetPosition HTTP/1.1 Accept-Language: pl-PL*

*User-Agent: Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; WOW64; Trident/5.0)*

*Content-Type: text/plain Accept-Encoding: text/html Host: 198.51.100.0:80*

*Connection: Keep-Alive Cache-Control: no-cache* ***Hard Deadline: 5***

In the above example *GetPosition* is a servlet that has to send a response during 5 seconds after receiving the request. First, the server tries to schedule the *GetPosition* in such a way, that the servlet will finish its execution before the deadline. If it will be possible then the servlet will send the response 220 as a result of this request. A sample response may have the following form (the message body contains information about the current and next positions):

## HTTP/1.1 220 Constraint Satisfied

*Date: So, 22 lis 2014 20:01:05 CET*

*Content-Length: 87 Content-Encoding: aslam Connection: close*

*Content-Type: text/html; charset=UTF-8 Server: HunterServer*

*<html> <body> current: Cracow, Warszawska: 45, change: Cracow, Szlak: 12</body> </html>*

If it will not be possible to find the feasible schedule, then the server will send the response 120, then the request is discarded. A sample response may be as follows:

## HTTP/1.1 120 Server Timeout

*Date: So, 22 lis 2014 20:21:55 CET*

*Content-Length: 42 Content-Encoding: aslam Connection: close*

*Content-Type: text/html; charset=UTF-8 Server: HunterServer*

*<html> <body>Server timeout</body> </html>*

1. *Header "Soft Deadline"*

Header ―Soft Deadline‖ also may be used only in requests. It defines the maximal time for processing the request, this deadline may be violated, but this results in degraded quality of service. For the same requests both soft and hard deadlines may be given. Server returns

response 220 even when the soft deadline is violated. The

―Soft Deadline‖ request header is defined as follows:

*Soft-Deadline :=* **Soft Deadline:** SP *time[***ms***]* CRLF Assume that the most precise results will be obtained if

the response will be sent not later than after 2s. Hence,

the request referring to the *GetPosition* additionally specifies a soft deadline. The request may have the following form:

*POST /task/ GetPosition HTTP/1.1 Accept-Language: pl-PL*

*User-Agent: Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; WOW64; Trident/5.0)*

*Content-Type: text/plain Accept-Encoding: text/html Host: 198.51.100.0:80*

*Connection: Keep-Alive Cache-Control: no-cache* ***Soft Deadline: 2***

## Hard Deadline: 5

If it will be not possible to schedule *GetPosition* before the hard deadline then the response 120 will be sent. Otherwise, the response 220 will be sent, even when the soft deadline will be violated. The servlet may take into consideration this delay to produce appropriate results,

e.g. it may modify the next position taking into consideration this delay:

## HTTP/1.1 220 Constraint Satisfied

*Date: So, 22 lis 2014 20:01:05 CET*

*Content-Length: 87 Content-Encoding: aslam Connection: close*

*Content-Type: text/html; charset=UTF-8 Server: HunterServer*

*<html> <body> current: Cracow, Warszawska: 45, change: Cracow, Szlak: 18 </body> </html>*

1. *Header "Remaining Time"*

Header ―Remaining Time‖ is specified in responses that are sent as a result of processing real-time requests. It specifies the remaining time that may be used for sending the response to the client. This information may be useful for proxies or gateways. The ―Remaining Time‖ response header is defined as follows:

*Remaining-Time :=* **Remaining Time:** SP *time[***ms***]*

CRLF

Fig. 1 presents the processing of deadlines by a proxy, a gateway and a server. First, the request R1 with deadline 5000 ms, is sent, we assume that the proxy processes this request during 200 ms, thus the deadline is modified. Next, the request is processed by the gateway, it will take 300 ms. After processing the request by the server that takes 4000 ms, the response 220 is sent to the server. Times of processing the response by the gateway

**4** Embedded Real-Time HTTP Server

and the proxy are equal 200 ms. Hence, the deadline for processing the request is not violated. In the second case, the request R2 is processed in time by the proxy, the

gateway and the server, but the proxy is not able to process the response and the deadline was exceeded by 100 ms, therefore the response code was changed to 120.

Fig. 1. Reduction of deadlines by intermediaries

R1

d=4500

R1

d=5000

R1

d=4800

220

d=100

220

d=300

220

d=500

R2

d=3500

R2

d=4000

R2

d=3800

120

d=100

220

d=100

220

d=300

Client

Proxy

Gateway

Server

Assume that the proxy is used for communication between a server and a client. To take into consideration delays caused by processing requests by the proxy, the server should use the ―Remaining Time‖ header. Assume that the processing of a request was finished 100 ms before the hard deadline, then the server should send the following response:

## HTTP/1.1 220 Constraint Satisfied

*Date: So, 22 lis 2014 20:31:12 CET*

*Content-Length: 57 Content-Encoding: aslam Connection: close*

*Content-Type: text/html; charset=UTF-8 Server: HunterServer*

## Remaining Time: 100 ms

*<html> <body> current: Cracow, Warszawska: 45, change: Cracow, Szlak: 12</body> </html>*

If the proxy is not able to process the response during 100 ms, then it should change the response to 120. A sample response may be the following:

## HTTP/1.1 120 Server Timeout

*Date: So, 22 lis 2014 20:31:54 CET*

*Content-Length: 40 Content-Encoding: aslam Connection: close*

*Content-Type: text/html; charset=UTF-8 Server: HunterServer*

*<html> <body>Proxy timeout</body> </html>*

Response 420 is sent when the value of the deadline is not valid, e.g. is a negative number or it is shorter than the time of processing the request by a server.

Response 520 is sent by servers which do not support real-time requests. The server may attach the proper message body, but it is not guaranteed that real-time constraints are satisfied.

* 1. ARCHITECTURE OF THE REAL-TIME EMBEDDED WEB SERVER

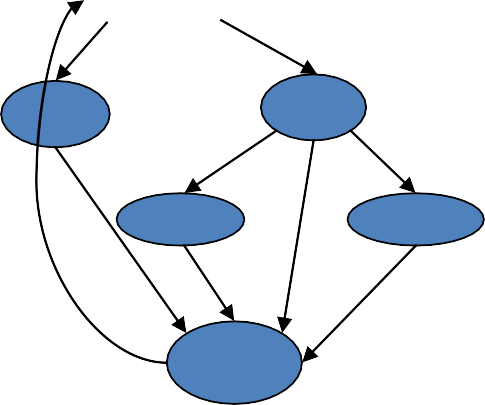
1. *Single-thread and Multi-thread Architectures*

Architectures of the EWS and traditional web servers are similar. In both cases a server may be single or multi- threaded. But the multithreading enables simultaneous processing of multiple requests. Fig. 2 presents the architecture of single- and multiple-threaded web servers. The specification of the embedded HTTP server in the form of a task graph consists of the following tasks:

* + **NextRequest** is the task that waits for the incoming requests appearing in HTTP socket. All received requests are stored in two queues, (described in the next subsection), where they are scheduled and then they are passed to task *ProcessReqest*. Task *NextRequest* is also responsible for scheduling the responses. The following responses are passed to task *Transmit*.
  + **ProcessReqest** is the task that is activated whenever next request is ready for processing. It gets the next request, removes it from the queue, and processes it. Requests GET and POST are passes to tasks *HandleGet* or *HandlePost* for futher processing.
  + **HandleGet** processes GET methods. Usually this task is implemented as servlet. The result is passed to task *ManageConnection* for completion of the final response.

Embedded Real-Time HTTP Server **5**

* + **HandlePost** processes POST methods. It is very similar to the *HandleGet* task.
  + **ManageConnection** is the task which prepares the response for clients and defines the status of connection. In the case where the connection is timed out, the deadline is exceeded or if an error occurred while processing the command, buffers are reset and the connection is closed.
  + **Transmit** is the process that transmits responses as HTML messages.



**NextRequest**

**Transmit**

**Process**

**Request**

**Req=GET**

**Req=POST**

**HandleGet**

**HandlePost**

**Manage**

**Connection**

(a)



**NextConnection**

**NextRequest**

**NextRequest**

**Transmit**

**Process**

**Request**

**Transmit**

**Process**

**Request**

**HandlePost**

**HandleGet**

**HandlePost**

**HandleGet**

**Manage**

**Connection**

**Manage**

**Connection**

(b)

Fig. 2 Architecture of a single-thread (a) and multiple-thread (b) web server

Single-thread server (Fig. 2a) processes all requests sequentially. Next request can not be served until the server will finish processing the previous one. The efficiency of single-thread server may be increased by applying a pipelined or parallel architecture. In the multi- threaded server (Fig. 2b) for each client another thread is created (process NextConnection). Each thread processes requests sent by another client. When the connection is closed, then the corresponding thread is destroyed.

Regardless of the server architecture all requests are processed in the FIFO order. Therefore, in the case of a large number of simultaneous requests some clients may wait a long time for the response. When EWS supports services for IoT systems, the web application expects the response during the required time period. In some cases the time may be critical due to short period of the data validity e.g. when data contains the position of the mobile system. While in other cases this period may be longer. Thus, more suitable would be the scheduling of requests according to their deadlines.

One of the most important requirements for the EWS is a small memory footprint and low memory requirements. Therefore, single-threaded EWS, or at least server with limited number of threads, is more appropriate. In case of heavy duty EWSs multi-core embedded processors may be used to increase the sever throughput.

1. *Request Scheduling*

We assume that the times required to perform tasks, corresponding to processing the requests, are known. Therefore, scheduling method based on Least Laxity First (LLF) [24] may be applied. Our Real-Time Embedded Web Server (RTEWS) accepts requests defined by the extended HTTP protocol described in p. III.

First, the algorithm schedules requests containing real- time requirements using LLF method. Other requests are served meanwhile i.e. when there are no real-time requests. FIFO scheduling is used for this purpose. Scheduling is performed by the NextRequest process (Fig. 2). Since, requests may feed the server continuously, the process modifies makespan after receiving each request. All requests are scheduled in two queues: LLFQ and FIFOQ. Process ProcessRequest gets the next request from the LLFQ, or if this queue is empty, then from the FIFOQ, and sends it to the appropriate task for processing.

**//** *algorithm starts each time when a new request* Rn

//*appears in time t.*

**Schedule(**Rn, t)

**if (**deadline(Rn)!=0) **then**

laxity(Rn)= tmax(Rn)-tproc(Rn);

add Rn to LLFQ in the order of ascending *laxity*

taking R with hard deadlines first;

**for** each Ri with hard deadline in LLFQ **do**

find processor *P* with min(tfinish(P));

**if** (tfinish(P)+ tproc(Ri)- tarr(Ri)<= tmax(Ri) **then**

tstart(P,Ri)= tfinish(P); tfinish(P)= tfinish(P)+ tproc(Ri);

**else**

**if (**Ri has hard deadline) send response 120; remove Ri from LLFQ;

**else //**soft deadline tstart(P,Ri)= tfinish(P); tfinish(P)= tfinish(P)+ tproc(Ri);

**else** add Ri to FIFOQ;

Fig. 3 Request scheduling algorithm

The draft of the scheduling algorithm is shown on Figure 3. *Rn* means the next request that arrived at time *t*,

**6** Embedded Real-Time HTTP Server

*tmax(*Ri*)* is the deadline specified for request *Ri*, *tproc(Ri)* is the estimated time of processing the request *Ri*, *tfinish(P)* is the finish time of all tasks currently assigned to processor *P*, *tarr(Ri)* is the time when request *Ri* arrived, *tstart(P,Ri)* denotes the time when the processor *P* will start to process request *Ri*.

Since during processing the next real-time requests

may arrive, the scheduler modifies schedule according to the LLF method. For each task the *laxity= tmax-tproc* is computed. All tasks are scheduled according to the *laxity* (task with the lowest laxity is scheduled first). If for the given task it is not possible to find the feasible schedule, then the server sends the response 120 and the request is canceled.

Requests containing soft deadlines are scheduled in the second pass. The scheduler modifies the LLFQ by adding these requests. First it tries to find the schedule that satisfies soft deadline, if it is not possible, then the schedule such that the time exceed is minimal is chosen.

1. *Example*

The scheduling of requests in the RTEWS architecture will be illustrated with an example. Assume that the server may serve four types of requests, processing of each request is performed by another task (e.g. servlets). Average execution times of all tasks are shown in Table 1. It is assumed that clients send requests using standard

methods POST and GET. Real-time requests will be denoted as POSTRT and GETRT. Let the scenario of arriving requests will be as shown in Table 2. The following columns contain: time of the request, request identifier, type of request, deadline tmax (for RT), constraint type (hard or soft), the task that is to be launched, the latency (tmax-tproc).

Table 1. Average Execution Time Of Tasks By RTEWS

|  |  |
| --- | --- |
| **Task Name** | **Average execution time [ms]** |
| T1 | 700 |
| T2 | 1200 |
| T3 | 1000 |
| T4 | 300 |

The makespan is shown in Figure 4. Arrows below the Gantt chart denote the events, corresponding to requests which arrived at RTEWS. Arrows on top of the chart denote points in time where scheduling of newly arrived requests is performed. *Sch(Rx, Ry)* means the order in which requests *Rx*, *Ry* will be processed. We do not consider preemptive scheduling, hence newly arrived requests are scheduled after finishing the currently executed tasks, even if they have the higher priority then the request being processed.

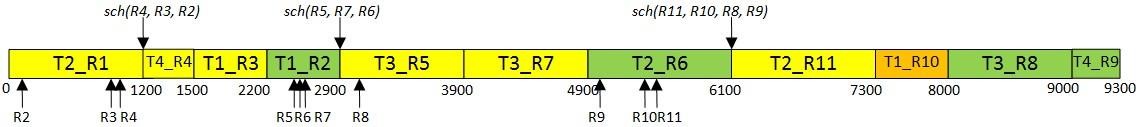


Fig. 4. The schedule of processing the requests in RTEWS

Real-time requests with imposed hard deadlines have the highest priority, thus they are scheduled first. If the deadline will not be satisfied, such request is not scheduled and it is missed. In the example, if request R10 had hard deadline, it would be missed, because it is not possible to find the proper schedule, in the best case it will exceed the deadline by 200ms. But actually, R10 has a soft deadline, thus it will be scheduled after request R11, which has a hard deadline. Requests that have no deadlines specified, have to wait for the execution of all handlers serving real-time requests. Therefore, the order of processing the requests is as follows. The R1 request has been received as first, at this time the server does not

process any other requests, so the task T2 was launched immediately as the handler of this request. Next, requests R2, R3 and R4 arrive during the execution of T2, thus they are scheduled after finishing the T1. The order is R4, R3 and R2, because R4 has more stringent time constraint,

i.e. treq(R4) + laxity(R4) < treq(R3) + laxity(R3) and R2 has no time constraints. The task serving request R2 is scheduled to execute after the processing of R3 and R4 will be finished, unless other real-time events will appear during this time. During the executing of T1, requests R5, R6, R7 arrived, they were scheduled according to their priorities. Request R8 arrived during

Table 2. Parameters of Requests Sent from the Client to the RT Server

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Request arrive time [ms]** | **Request ID** | **Request type** | **Deadline [ms]** | **Constraint type** | **Task** | **Latency [ms]** |
| 0 | R1 | POSTRT | 2000 | hard | T2 | 800 |
| 100 | R2 | POST | - | - | T1 | - |
| 1100 | R3 | POSTRT | 3000 | hard | T1 | 2300 |
| 1120 | R4 | GETRT | 2500 | hard | T4 | 2200 |
| 2500 | R5 | GETRT | 3000 | hard | T3 | 2400 |
| 2600 | R6 | GET | - | - | T2 | - |
| 2610 | R7 | POSTRT | 3000 | hard | T3 | 2400 |
| 3000 | R8 | POST | - | - | T3 | - |
| 5000 | R9 | POST | - | - | T4 | - |
| 5300 | R10 | POSTRT | 1300 | soft | T1 | 600 |
| 5500 | R11 | GETRT | 2000 | hard | T2 | 800 |

Embedded Real-Time HTTP Server **7**

execution of task T3, but since this is no real-time request, there is no necessity for rescheduling after finishing T3, thus request R8 is simply added to the FIFOQ (after R6). Requests R8 and R9 were scheduled at the end, in order of coming, because they do not have a time constraint in the header, and R11 and R10 had to be executed first to satisfy their deadlines. Finally, our server processes all requests in the order that guarantees satisfying all hard real-time requests. It should be noticed that in case of FIFO scheduling, deadlines would not be satisfied for requests R7 and R11.

* 1. EXPERIMENTAL RESULTS

In order to demonstrate the practical benefits of using RTEWS for processing real-time requests, we have developed a lightweight server. Only methods GET and POST and headers defining hard deadlines were available. RTEWS was implemented in C++, in the MicroC/OS-II environment running on the Nios II processor built in Altera Cyclone II FPGA. Next, we performed experiments showing the effectiveness of request processing. The server was tested with increasing number of incoming requests per second. The test was performed in two passes. First, all requests were precessed in the FIFO order, like in existing web servers. During the second pass, our scheduling strategy was applied. The number of concurrent requests has been increased from 10 to 3000. For each case the number of canceled real- time requests were analyzed and counted. The results are presented in Figure 5. The x-axis shows the number of incoming requests and the y-axis the percentage of RT requests canceled by the server, because they do not meet deadlines. Server executed a few different tasks as handlers of the requests. The average execution time of these tasks was in the range from 100 to 3000 ms.The deadlines for RT requests were not higher than 3500 ms.

It may be observed that the number of missed real-time requests, obtained in the HTTP server was a few times larger than in the RTEWS. The number of missed RT requests in RTEWS did not exceed 5%, even for the big number of concurrent requests. For HTTP server the RT requests did not meet the time requirements in 40% of such requests. For high number of concurrent requests this percentage increased up to 80-90%.

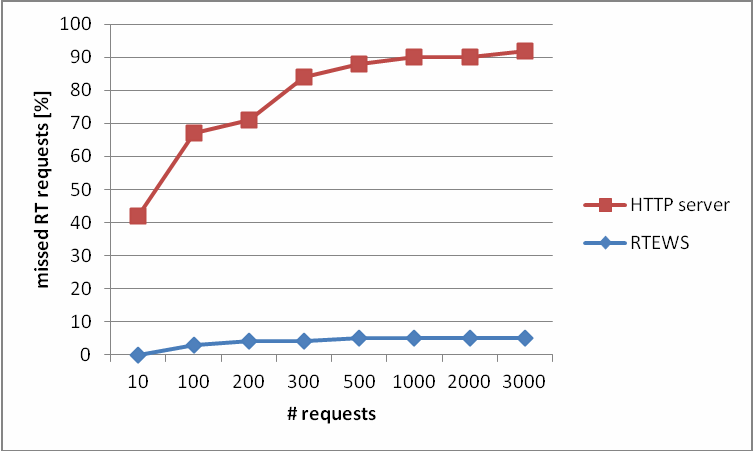


Fig. 5. The number of missed real-time requests.

* 1. CONCLUSIONS

In this paper the architecture of embedded real-time web server was proposed. The server schedules HTTP requests taking into consideration hard and soft deadlines as well as the expected server load. Specifications of deadlines are possible by using the proposed extensions to the HTTP protocol. LLF scheduling of request handlers guarantees finding the proper schedule (if exists) and the high quality of service. If it is not possible to meet the deadline, the server cancels the request and sends the appropriate response to the client.

Experimental results showed that the number of missed real-time requests in the proposed server is a few times smaller than using the existing HTTP servers. Since in our approach requests that cannot be served in expected time period are canceled, thus the overloading or falling over of the server is avoided.

The server may be very useful in IoT applications that use embedded systems according to the Sensing as a Service business model. We believe that in the near future more and more embedded systems equipped with different sensors and located in different places will support real-time services for different web applications.

REFERENCES

[1] Atzoria, L., Iera, A., Morabito, G.: The Internet of Things: A survey. Computer Networks, vol. 54, no.15, pp. 2787- 2805 (2010).

[2] Höller, V. Tsiatsis, C. Mulligan, S. Karnouskos, S. Avesand, D. Boyle: From Machine-to-Machine to the Internet of Things: Introduction to a New Age of Intelligence. Elsevier, 2014.

[3] Xiang Sheng; Jian Tang; Xuejie Xiao; Guoliang Xue, "Sensing as a Service: Challenges, Solutions and Future Directions‖, *IEEE Sensors Journal,* vol.13, no.10, pp.3733-3741, Oct. 2013.

[4] Dargie, W. and Poellabauer, C., "Fundamentals of wireless sensor networks: theory and practice", John Wiley and Sons, 2010.

[5] R. Fielding, J. C. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee, ‖Hypertext Transfer Protocol - HTTP/1.1‖, IETF, RFC 2616, 1999.

[6] Guinard, D., Trifa, V., Mattern, F., Wilde, E.: From the Internet of Things to the Web of Things: Resource Oriented Architecture and Best Practices. In: Uckelmann, Dieter; Harrison, Mark; Michahelles, Florian (Eds.) Architecting the Internet of Things, Springer-Verlag Berlin Heidelberg (2011).

[7] H. Schulzrinne, A. Rao, R. Lanphier, *Real Time Streaming Protocol (RTSP)*, IETF, RFC 2326,1998

[8] Peter Saint-Andre, Kevin Smith, Remko TronconPeter Saint-Andre, Kevin Smith, Remko Troncon, XMPP: The Definitive Guide, [O'Reilly Media,](http://it-ebooks.info/publisher/3/) 2009.

[9] Crane, Dave; McCarthy, Phil: Comet and Reverse Ajax: The Next-Generation Ajax 2.0. Apress, (2008).

[10] F. M. Aymerich, G. Fenu, S. Surcis, ―A real time financial system based on grid and cloud computing‖ *ACM symposium on Applied Computing*, March 2009, New York, pp 1219–1220.

[11] S. Liu, G. Quan, S. Ren, ―On-Line Scheduling of Real- Time Services for Cloud Computing‖ *World Congress on Services*, July 2010, Miami, pp 459–464.

**8** Embedded Real-Time HTTP Server

[12] W. Tsai, Q. Shao, X. Sun, J. Elston, ―Real-Time Service- Oriented Cloud Computing‖ *World Congress on Services*, July 2010, Miami, pp 473–478.

[13] K. H. Kim, A. Beloglazov, R. Buyya, ―Power-aware provisioning of cloud resources for realtime services‖ *International Workshop on Middleware for Grids, Clouds and e-Science*, 2009, New York.

[14] K. Kumar, J. Feng, Y. Nimmagadda, Y. Lu, ―Resource Allocation for Real-Time Tasks using Cloud Computing‖ *International Conference on Computer Communications and Networks (ICCCN)*, July 2011, pp. 1-7.

[15] S. Bąk, R.Czarnecki, S. Deniziak, "Synthesis of Real-

**Authors’ Profiles**



Zielona Góra.

**Radosław Czarnecki** is Assistant Professor in Division of Computer Science in Faculty of Electrical and Computer Engineering, Cracow University of Technology, Poland. In 2002 he received MSc in Electrical Engineering from Cracow University of Technology, and in 2008 PhD degree in Computer Science from University of

Time Applications for Internet of Things", LNCS vol.

7719, pp. 37-51, 2013.

[16] S. Bąk, R. Czarnecki, S. Deniziak "Synthesis of real-time cloud applications for Internet of things" *Turkish Journal of Electrical Engineering and Computer Sciences*, to be published, [http://dx.doi.org/10.3906/elk-1302-178.](http://dx.doi.org/10.3906/elk-1302-178)

[17] Deniziak, S.; Bak, S., "Synthesis of real time distributed applications for cloud computing", Proc of the IEEE *Federated Conference on Computer Science and Information Systems (FedCSIS),* pp.743-752, 2014.

[18] Embedthis Software, Appweb Architecture, [http://appwebserver.org/products/appweb/doc/ref/architect](http://appwebserver.org/products/appweb/doc/ref/architecture.html) [ure.html,](http://appwebserver.org/products/appweb/doc/ref/architecture.html) 2014.

[19] Unicoi Systems, Fusion Embedded™ HTTP, [http://www.unicoi.com/product\_briefs/http.pdf,](http://www.unicoi.com/product_briefs/http.pdf) 2013.

[20] Real Time Logic, Barracuda Web Server, [https://realtimelogic.com/products/barracuda-web-server/,](https://realtimelogic.com/products/barracuda-web-server/) 2014.

[21] Duquennoy, S.; Grimaud, G.; Vandewalle, J.-J., "Smews: Smart and Mobile Embedded Web Server", *International Conference on Complex, Intelligent and Software Intensive Systems, 2009. CISIS '09,* pp.571-576, 2009.

[22] Trotter Cashion, Introducing Chloe - The Realtime Web Server, [http://www.trottercashion.com/2011/06/13/intro-](http://www.trottercashion.com/2011/06/13/intro-ducing-chloe.html) [ducing-chloe.html,](http://www.trottercashion.com/2011/06/13/intro-ducing-chloe.html) 2011.

[23] S. Loreto, M. Thomson, G. Wilkins, ―Hypertext Transfer Protocol (HTTP) Timeouts‖, IETF, draft-loreto-http- timeout-00, 2010.

[24] G. C. Buttazzo *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*, Kluwer Academic Publishers, 1997.

The main profile of his research focuses on methodologies of designing a real-time Internet of Things systems. Previous researches were directed at synthesis methods for hardware- software distributed systems and reconfigurable embedded systems. He regular published scientific papers in embedded systems and synthesis methods of such systems in journals and conference papers and presented them on national and international meetings to promote the research. He is the author of two papers on the synthesis of real-time applications for Internet of Things in IEEE journals.

**Stanisław Deniziak** is Professor of Computer Science in Department of Computer Science, Kielce University of Technology, Poland. He received MSc in Computer Science from Warsaw University of Technology, and PhD degree from Gdańsk University of Technology. In 2006 he received DSc in Computer Science from Warsaw

University of Technology. Now, he is Vice Dean for Research and Promotion of Faculty of Electrical Engineering, Automatics and Computer Science, Kielce University of Technology.

He has published 78 research papers in various international and national journals and conferences. He is active reviewer end editorial member of 7 international journals such Journal of Systems and Software, Computing, Microprocessors and Microsystems, International Journal of Applied Mathematics and Computer Science, The Open Cybernetics & Systemic Journal, International Journal of the Physical Sciences, Annales UMCS - Sectio A Informatica. He has reviewed research papers of many international conferences like: IEEE Design Automation Conference, International Conference of Computational Methods in Sciences and Engineering etc.

Prof. Deniziak is IEEE and IEEE Computer Society Member.

**How to cite this paper:** Radosław Czarnecki, Stanislaw Deniziak,"Embedded Real-Time HTTP Server", IJCNIS, vol.7, no.5, pp.1-8, 2015.DOI: 10.5815/ijcnis.2015.05.01

1. ***J. Computer Network and Information Security,* 2015, 5, 1-8** Published Online April 2015 in MECS (<http://www.mecs-press.org/)> DOI: 10.5815/ijcnis.2015.05.01

# 嵌入式实时HTTP服务器

# Radosław Czarnecki

Division of Computer Science, Cracow University of Technology, Poland Email: [czarneck@pk.edu.pl](mailto:czarneck@pk.edu.pl)

# Stanislaw Deniziak

Department of Computer Science, Kielce University of Technology, Poland Email: [s.deniziak@computer.org](mailto:s.deniziak@computer.org)

***摘要***—本文介绍了嵌入式实时Web服务器的架构。与现有的Web服务器不同，在我们的方法中，请求不是按照“先进先出”顺序处理，而是根据它们的截止日期和预期的服务器负载进行处理。为此，使用最小松弛优先调度方法。首先，提供具有强制实时约束的请求。然后处理由软期限包围的请求。最后，没有时间要求的请求按照他们到达的顺序来服务。我们还提供对超文本传输协议的实时扩展。我们提出了标头，用于定义硬和软的截止期限，以及包含时间信息的响应，这些响应发送到客户端应用程序。实验结果表明，在实时应用的情况下，我们的服务器错过非常少的请求，由于超时，然后现有的解决方案。所呈现的服务器对于实现由嵌入式系统支持的实时服务可能是非常有用的。在未来的实时“物联网”应用。

***关键词*—**Web服务器，嵌入式系统，实时系统，HTTP，物联网，传感作为服务。

* 1. 引言

越来越多的嵌入式系统连接到互联网。新兴的概念，如物联网（IoT）[1]，机器对机器通信（M2M）[2]，传感作为服务（S2aaS）[3]，无线传感器网络[4]启用Web的设备。这样的设备可以使用HTTP协议与Web应用程序通信[5]，其中许多都是由Web浏览器控制的。为此，应该内置嵌入式Web服务器（EWS）。EWS通常是一个轻量级应用程序，仅实现由HTTP协议定义的主要方法。它包含简单的网页，用于交互的表单。 EWS用于控制网络打印机，无线路由器，网络摄像机和其他设备或传感器。预期在几年中，几乎每个产品可以使用无线通信方法在因特网上被识别和跟踪。此外，他们中的很多将向Web应用程序提供传感数据。

现有的基于网络的传感应用不考虑时间要求。虽然增加感测服务的数量将导致具有内置EWS的传感器将面临实时条件，这应当满足以提供适当和所需的水平服务质量（QoS）。最近发现未来的物联网或面向服务的架构应该解决实时方面。一些Web应用程序有时间要求，特别是在环境监测，运输等领域。物联网将包括数十亿的智能通信或互联网连接对象（ICO），它们将具有感应，致动和数据处理能力。每个ICO将具有一个或多个嵌入式传感器，其将捕获潜在的大量数据。为了能够处理大量的请求，这种ICO应该考虑实时约束。现有的基于网络的传感应用不考虑时间要求。虽然增加感测服务的数量将导致具有内置EWS的传感器将面临实时条件，这应当满足以提供适当和所需的水平服务质量（QoS）。最近发现未来的物联网或面向服务的架构应该解决实时方面。一些Web应用程序有时间要求，特别是在环境监测，运输等领域。物联网将包括数十亿的智能通信或互联网连接对象（ICO），它们将具有感应，致动和数据处理能力。每个ICO将具有一个或多个嵌入式传感器，其将捕获潜在的大量数据。为了能够处理大量的请求，这种ICO应该考虑实时约束。

实时应用程序期望传感器或外部服务在可预测的时间段内做出响应。不幸的是，HTTP不支持实时约束。当在预期时间内未收到对请求的响应时，无法确定是否是由EWS超载，网络故障，服务器故障或繁重的网络流量造成的。此外，在现有的HTTP服务器中，不能控制进入请求的处理顺序或预测处理时间。在实时环境中，EWS应该首先根据截止日期处理请求，其次，如果它不能在预期的时间内处理请求，EWS应该向客户端应用程序发送超时消息。满足上述要求的服务器将显着提高实时IoT系统中的QoS，特别是当ICO支持大量实时网络应用的感测数据时，并且当获得这些数据需要耗时的计算时。

在这项工作中，我们介绍嵌入式实时HTTP服务器的架构。为了实现对HTTP请求的实时要求，我们将定义HTTP协议的实时扩展。服务器根据其优先级（基于实时要求）和预期处理时间来调度所有请求。我们的服务器的主要设备将是嵌入式系统实时支持传感数据。根据我们最好的知识，没有类似的嵌入式系统解决方案。

本文的其余部分安排如下。 在下一节中介绍了相关的作品。 在第3节中，我们提供了HTTP协议的实时扩展。第4节描述了实时HTTP服务器的架构。 在第5节中给出了显示在实时网络应用中应用我们的服务器的优点的实验结果。最后，在第6节中，我们得出一些结论，并且展示在该领域中进一步开发的工作。

* 1. 相关工作

应用程序期望传感器或外部服务在可预测的时间段内做出响应。不幸的是，HTTP不支持实时约束。当在预期时间内未收到对请求的响应时，无法确定是否是由EWS超载，网络故障，服务器故障或繁重的网络流量造成的。此外，在现有的HTTP服务器中，不能控制进入请求的处理顺序或预测处理时间。在实时环境中，EWS应该首先根据截止日期处理请求，其次，如果它不能在预期的时间内处理请求，EWS应该向客户端应用程序发送超时消息。满足上述要求的服务器将显着提高实时IoT系统中的QoS，特别是当ICO支持大量实时网络应用的感测数据时，并且当获得这些数据需要耗时的计算时。

网络应用程序通常在云计算的上下文中考虑。关于实时云计算（RTCC）的当前工作主要集中在两个领域：将现有的网络技术应用于这种新的范例并且开发用于实时应用的软件架构。最近对实时任务的资源分配进行了研究。 Aymerich 等人 [10]为基于云计算技术的实时金融系统开发了基础设施。 Liu 等人 [11]展示了如何用不同的效用函数调度实时任务。实时任务被调度为非抢占式，目的是通过使用时间效用函数（TUF）来最大化总效用。 Tsai 等人 [12]讨论了云基础设施上的实时数据库分区。 Kim 等人 [13]研究实时云服务的资源的功率感知供应。在他们的工作中，实时约束在客户和云提供商之间的服务水平协议（SLA）中指定。 SLA指定协商的协议，包括QoS。在这种云模型中，服务提供商负责分配资源。他们的工作检查电力管理，而资源分配应符合SLA。上述研究都没有考虑从云中可用的一组不同类型的资源中选择实现成本效益的实时任务。 Kumar 等人 [14]开发了一种用于具有实时任务的应用的资源分配算法。他们提出了一种考虑时间重叠以有效分配资源的EDF-greedy方案。 IoT的实时云应用的成本感知合成方法在[15] [16] [17]中提出。

嵌入式Web服务器有很多实现。 Appweb [18]是一个紧凑的多线程服务器，支持ESP，Ejscript和PHP框架的内存模块。 Fusion Embedded HTTP Server [19]只支持GET和POST方法。它可以处理多个并发请求，主要优势是非常小的内存占用。梭子鱼Web服务器[20]是一个工业强度，小嵌入Web服务器引擎，针对紧凑，深度嵌入式设备进行了优化。 Smews [21]是WoT系统的非常高效和非常小的web服务器的原型。

没有考虑实时约束的已知Web服务器。 被称为实时网络服务器的Chloe [22]处理“实时网络”，即使得浏览器能够在其作者发布时立即接收信息的解决方案。 实时网络与实时计算基本上不同，因为不知道什么时候或者是否将接收到响应。

* 1. 实时HTTP

HTTP是用于客户端和服务器之间的通信的无状态协议[Fie99]。 HTTP会话由一系列请求 - 响应事务组成。客户端应用程序发送请求，该请求可以对应于以下方法之一：GET，HEAD，POST，PUT，DELETE，OPTIONS，TRACE和CONNECT。作为响应，服务器应用发回包含状态行和所请求资源的消息。并非所有方法都需要在服务器应用程序中实现。嵌入式系统通常使用轻量级Web服务器，其中最重要的是占用内存很少，CPU占用率最低和较高可靠性。

虽然提出了一些用于规定时间要求的扩展[23]，但HTTP协议不考虑实时要求。请求按FIFO顺序处理，而不考虑预期的处理时间。因此，不可能预测客户端何时将接收响应，以及不可能定义与请求相关联的时间约束。服务器延迟取决于必须处理的请求数和处理时间。

为了能够指定实时约束，我们提出对HTTP协议（RT-HTTP）的实时扩展。 HTTP / 1.1中定义的所有方法都在RT-HTTP中可用，此外，三个新标题（硬截止期限，软截止时间和剩余时间）以及四个新响应（120 Server Timeout，220 Constraint Satisfied，420 Wrong Deadline， 520期限不支持）。

1. *标题“硬截止日期”*

头---限制期限“定义了硬时间约束，即服务器处理请求的最大时间。此标头只能在请求中指定。不能违反硬约束，否则服务器应返回响应120（服务器超时）。当请求被及时处理时，服务器向客户端发送220（Constraint Satisfied）响应。

“请求最后期限”请求头在RT-HTTP中定义如下：

Hard-Deadline：= Hard截止日期：SP时间[ms] CRLF

其中：SP是空格字符，CRLF表示结束行，时间表示以秒（默认）或毫秒给出的截止时间值。

假设物联网应用跟踪公共交通系统中的公共汽车位置。 每个总线配备有GPS和EWS，支持当前总线位置。 为了获得最新的公交车位置，应及时提供信息。 下面显示了包含硬性截止日期的示例POST请求：

*POST /task/GetPosition HTTP/1.1*

*Accept-Language: pl-PL*

*User-Agent: Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; WOW64; Trident/5.0)*

*Content-Type: text/plain*

*Accept-Encoding: text/html*

*Host: 198.51.100.0:80*

*Connection: Keep-Alive*

*Cache-Control: no-cache*

***Hard Deadline: 5***

在上面的示例中，GetPosition是一个servlet（Server Applet），它必须在接收到请求后5秒内发送响应。 首先，服务器尝试以这样的方式调度GetPosition，即servlet将在截止日期之前完成其执行。 如果这将是可能的，那么作为该请求的结果，小服务器将发送响应220。 示例响应可以具有以下形式（消息主体包含关于当前位置和下一位置的信息）：

***HTTP/1.1 220 Constraint Satisfied***

*Date: So, 22 lis 2014 20:01:05 CET*

*Content-Length: 87*

*Content-Encoding: aslam*

*Connection: close*

*Content-Type: text/html; charset=UTF-8*

*Server: HunterServer*

*<html> <body> current: Cracow, Warszawska: 45, change: Cracow, Szlak: 12</body> </html>*

如果不可能找到可行的调度，则服务器将发送响应120，然后丢弃该请求。 样本响应可以如下：

***HTTP/1.1 120 Server Timeout***

*Date: So, 22 lis 2014 20:21:55 CET*

*Content-Length: 42*

*Content-Encoding: aslam*

*Connection: close*

*Content-Type: text/html; charset=UTF-8*

*Server: HunterServer*

*<html> <body>Server timeout</body> </html>*

当截止时间的值无效时，发送响应420。 是负数或者比服务器处理请求的时间短。

1. *标题“软截止日期”*

标题 - 软截止日期“也可以只在请求中使用。 它定义了处理请求的最大时间，可能违反此最后期限，但这会导致服务质量下降。 对于相同的请求，可以给

出软和最终期限。 服务器返回即使当违反软期限时也响应220。 “软最后期限”请求头定义如下：

软最后期限：=软期限：SP时间[ms] CRLF

假设如果响应将在不迟于2s之后发送，则将获得最精确的结果。 因此，引用GetPosition的请求另外指定了软限期。 请求可以具有以下形式：

*POST /task/ GetPosition HTTP/1.1*

*Accept-Language: pl-PL*

*User-Agent: Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; WOW64; Trident/5.0)*

*Content-Type: text/plain*

*Accept-Encoding: text/html*

*Host: 198.51.100.0:80*

*Connection: Keep-Alive*

*Cache-Control: no-cache*

***Soft Deadline: 2***

***Hard Deadline: 5***

如果不可能在硬限制之前调度GetPosition，则将发送响应120。 否则，将发送响应220，即使当违反软最终期限时。 servlet可以考虑这个延迟以产生适当的结果，例如。 考虑到这个延迟，它可以修改下一个位置：

*HTTP/1.1 220 Constraint Satisfied*

*Date: So, 22 lis 2014 20:01:05 CET*

*Content-Length: 87*

*Content-Encoding: aslam*

*Connection: close*

*Content-Type: text/html; charset=UTF-8*

*Server: HunterServer*

*<html> <body> current: Cracow, Warszawska: 45, change: Cracow, Szlak: 18 </body> </html>*

1. *标题“剩余时间”*

在作为处理实时请求的结果发送的响应中指定报头 -剩余时间。 它指定可用于将响应发送到客户机的剩余时间。 此信息可能对代理或网关有用。 消息时间响应头定义如下：

剩余时间：=剩余时间：SP时间[ms] CRLF

图。 图1示出了代理，网关和服务器对最终期限的处理。 首先，发送具有期限5000ms的请求R1，我们假设代理在200ms期间处理该请求，因此修改期限。 接下来，请求由网关处理，它将需要300毫秒。 在服务器处理需要4000ms的请求之后，响应220被发送到服务器。网关和代理处理响应的时间等于200ms。 因此，不违反用于处理请求的最后期限。 在第二种情况下，请求R2由代理，网关和服务器及时处理，但是代理不能处理响应，并且超过100ms的期限，因此响应代码被改变为120。

R1

d=4500

R1

d=5000

R1

d=4800

220

d=100

220

d=300

220

d=500

R2

d=3500

R2

d=4000

R2

d=3800

120

d=100

220

d=100

220

d=300

客户端

代理

网关

服务

图 1. 减少中介耗费时间

假设代理用于服务器和客户端之间的通信。为了考虑由代理处理请求引起的延迟，服务器应该使用-RemainingTime‖头。

假设请求的处理在硬截止时间之前100毫秒完成，则服务器应发送以下响应：

***HTTP/1.1 220 Constraint Satisfied***

*Date: So, 22 lis 2014 20:31:12 CET*

*Content-Length: 57*

*Content-Encoding: aslam*

*Connection: close*

*Content-Type: text/html; charset=UTF-8*

*Server: HunterServer*

***Remaining Time: 100 ms***

*<html> <body> current: Cracow, Warszawska: 45, change: Cracow, Szlak: 12</body> </html>*

如果代理无法在100 ms内处理响应，那么它应该将响应更改为120.示例响应可能如下：

***HTTP/1.1 120 Server Timeout***

*Date: So, 22 lis 2014 20:31:54 CET*

*Content-Length: 40*

*Content-Encoding: aslam*

*Connection: close*

*Content-Type: text/html; charset=UTF-8*

*Server: HunterServer*

*<html> <body>Proxy timeout</body> </html>*

当截止时间的值无效时，发送响应420。 是负数或者比服务器处理请求的时间短。

响应520由不支持实时请求的服务器发送。 服务器可以附加适当的消息体，但不能保证满足实时约束。

* 1. 实时嵌入式WEB服务器的架构

1. 单线程和多线程架构

EWS的架构和传统的Web服务器是类似的。 在这两种情况下，服务器可以是单线程或多线程的。 但是多线程能够同时处理多个请求。 图。 图2给出了单线程和多线程Web服务器的架构。 任务图形式的嵌入式HTTP服务器的规范包括以下任务：

. NextRequest是等待在HTTP套接字中出现的传入请求的任务。 所有接收到的请求都存储在两个队列中（在下一小节中描述），在那里它们被调度，然后它们被传递给任务ProcessReqest。 任务NextRequest也负责调度响应。 以下响应传递给任务Transmit。

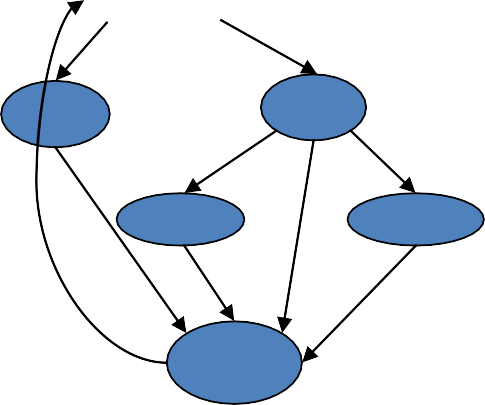
. ProcessReqest是在下一个请求准备好处理时激活的任务。 它获得下一个请求，从队列中删除它，并处理它。 请求GET和POST是传递给任务HandleGet或HandlePost以进行进一步处理。

. HandleGet进程GET方法。 通常这个任务被实现为servlet。 结果传递到任务ManageConnection以完成最终响应。

. HandlePost处理POST方法。 它非常类似于HandleGet任务

. ManageConnection是为客户端准备响应并定义连接状态的任务。 在连接超时的情况下，超过期限或者如果在处理命令时发生错误，则缓冲器被复位并且连接被关闭。

. Transmit是将响应作为HTML消息传输的过程。



**NextRequest**

**Transmit**

**Process**

**Request**

**Req=GET**

**Req=POST**

**HandleGet**

**HandlePost**

**Manage**

**Connection**

(a)



**NextConnection**

**NextRequest**

**NextRequest**

**Transmit**

**Process**

**Request**

**Transmit**

**Process**

**Request**

**HandlePost**

**HandleGet**

**HandlePost**

**HandleGet**

**Manage**

**Connection**

**Manage**

**Connection**

(b)

图. 2 单线程（a）和多线程（b）web服务器的体系结构

单线程服务器（图2a）顺序处理所有请求。 在服务器完成处理前一个请求之前，无法提供下一个请求。 可以通过应用流水线或并行架构来提高单线程服务器的效率。 在多线程服务器（图2b）中为每个客户端创建另一个线程（进程NextConnection）。 每个线程处理另一客户端发送的请求。 当连接关闭时，相应的线程被销毁。

无论服务器体系结构如何，所有请求都按FIFO顺序处理。 因此，在大量同时请求的情况下，一些客户端可能等待很长时间的响应。 当EWS支持物联网系统的服务时，Web应用程序期望在所需时间段内的响应。 在一些情况下，由于数据有效性的短周期，时间可能是关键的。 当数据包含移动系统的位置时。 而在其他情况下，这一时期可能更长。 因此，更合适的是根据它们的最后期限调度请求。

EWS最重要的要求之一是

小内存占用和低内存要求。 因此，单线程EWS或至少具有有限线程数的服务器是更合适的。 在重型EWS的情况下，多核嵌入式处理器可以用于增加服务器吞吐量。

1. 请求调度

我们假设执行与处理请求相对应的任务所需的时间是已知的。 因此，可以应用基于最小松弛优先（LLF）[24]的调度方法。 我们的实时嵌入式Web服务器（RTEWS）接受第44页中描述的扩展HTTP协议定义的请求。 III。

首先，算法使用LLF方法调度包含实时要求的请求。

同时提供其他请求，即当没有实时请求时。 FIFO调度用于此目的。 调度由NextRequest进程执行（图2）。 由于请求可以连续地馈送服务器，所以该过程在接收每个请求之后修改makepan。 所有请求都安排在两个队列中：LLFQ和FIFOQ。 Process ProcessRequest从LLFQ获取下一个请求，或者如果此队列为空，则从FIFOQ获取下一个请求，并将其发送到适当的任务进行处理。

tmax（Ri）是为请求Ri指定的最后期限，tproc（Ri）是处理请求Ri的估计时间，tfinish（P）是当前分配给处理器P的所有任务的完成时间，tarr（Ri）当请求Ri到达时，tstart（P，Ri）表示处理器P将开始处理请求Ri的时间。

由于在处理期间下一个实时请求可能到达，调度器根据LLF方法修改调度。对于每个任务，计算laxity = tmax-tproc。所有任务根据松弛度调度（具有最低松弛度的任务首先被调度）。如果对于给定任务不可能找到可行的调度，则服务器发送响应120并且取消请求。

包含软最后期限的请求将在第二阶段计划。调度器通过添加这些请求来修改LLFQ。首先，它试图找到满足软限期的调度，如果不可能，则选择时间超过最小的调度。

**//** *algorithm starts each time when a new request* Rn

//*appears in time t.*

**Schedule(**Rn, t)

**if (**deadline(Rn)!=0) **then**

laxity(Rn)= tmax(Rn)-tproc(Rn);

add Rn to LLFQ in the order of ascending *laxity*

taking R with hard deadlines first;

**for** each Ri with hard deadline in LLFQ **do**

find processor *P* with min(tfinish(P));

**if** (tfinish(P)+ tproc(Ri)- tarr(Ri)<= tmax(Ri) **then**

tstart(P,Ri)= tfinish(P); tfinish(P)= tfinish(P)+ tproc(Ri);

**else**

**if (**Ri has hard deadline) send response 120; remove Ri from LLFQ;

**else //**soft deadline tstart(P,Ri)= tfinish(P); tfinish(P)= tfinish(P)+ tproc(Ri);

**else** add Ri to FIFOQ;

图. 3 请求调度算法

1. *示例*

将用示例来说明RTEWS体系结构中的请求的调度。 假设服务器可以服务四种类型的请求，每个请求的处理由另一个任务（例如servlet）执行。 所有任务的平均执行时间如表1所示。假设客户端使用标准方法POST和GET发送请求。 实时请求将被表示为POSTRT和GETRT。 使得到达请求的场景将如表2所示。以下列包含：请求的时间，请求标识符，请求的类型，期限tmax（对于RT），约束类型（硬或软），任务是 待启动，等待时间（tmax-tproc）。

表1. 任务的平均执行时间由RTEWS

|  |  |
| --- | --- |
| **任务名称e** | **平均执行时间 [ms]** |
| T1 | 700 |
| T2 | 1200 |
| T3 | 1000 |
| T4 | 300 |

构建时间如图4所示。甘特图下方的箭头表示事件，对应于到达RTEWS的请求。 图表上方的箭头表示执行新到达请求的调度的时间点。 Sch（Rx，Ry）表示请求Rx，Ry将被处理的顺序。 我们不考虑抢先调度，因此在完成当前执行的任 务之后调度新到达的请求，即使它们具有比正在处理的请求更高的优先级。

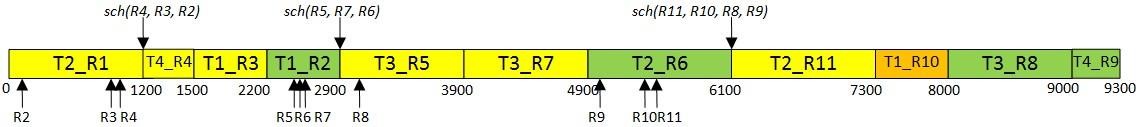


图. 4. 在RTEWS中处理请求的计划

具有强制的最后期限的实时请求具有最高优先级，因此它们被排在第一。如果不能满足最后期限，则不会安排这样的请求并且错过。在该示例中，如果请求R10具有硬的最后期限，则将被错过，因为不可能找

到适当的调度，在最好的情况下其将超过200ms的最后期限。但实际上，R10有一个软的最后期限，因此它将被安排在请求R11后，它有一个硬的最后期限。没有指定任何截止时间的请求必须等待所有处理器执行实时请求。

因此，处理请求的顺序如下。 R1请求已作为第一个接收，此时服务器不处理任何其他请求，因此任务T2立即作为此请求的处理程序启动。接下来，请求R2，R3和R4在T2的执行期间到达，因此它们在完成T1之后被调度。顺序是R4，R3和R2，因

为R4具有更严格的时间约束，即treq（R4）+ laxity

（R4）<treq（R3）+ laxity（R3），R2没有时间限制。

任务服务请求R2被调度为在R3和R4的处理完成之后执行，除非在该时间期间出现其他实时事件。在执行T1期间，请求R5，R6，R7到达，它们根据它们的优先级被调度。请求R8在任务T3的执行期间到达，但由于这不是实时请求，因此在完成T3之后不需要重新调度，因此请求R8被简单地添加到FIFOQ（在R6之后）。请求R8和R9按照来到的顺序安排在结束，因为它们在头部中没有时间约束，并且必须首先执行R11和R10以满足其最后期限。最后，我们的服务器以保证满足所有硬实时请求的顺序处理所有请求。应当注意，在FIFO调度的情况下，对于请求R7和R11将不满足最终期限。

表 2. 从客户端发送到RT服务器的请求的参数

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **请求到达的时间[ms]** | **请求编号** | **请求类型** | **最后期限 [ms]** | **约束类型** | **任务** | **潜在[ms]** |
| 0 | R1 | POSTRT | 2000 | hard | T2 | 800 |
| 100 | R2 | POST | - | - | T1 | - |
| 1100 | R3 | POSTRT | 3000 | hard | T1 | 2300 |
| 1120 | R4 | GETRT | 2500 | hard | T4 | 2200 |
| 2500 | R5 | GETRT | 3000 | hard | T3 | 2400 |
| 2600 | R6 | GET | - | - | T2 | - |
| 2610 | R7 | POSTRT | 3000 | hard | T3 | 2400 |
| 3000 | R8 | POST | - | - | T3 | - |
| 5000 | R9 | POST | - | - | T4 | - |
| 5300 | R10 | POSTRT | 1300 | soft | T1 | 600 |
| 5500 | R11 | GETRT | 2000 | hard | T2 | 800 |

* 1. 实验结果

为了演示使用RTEWS处理实时请求的实际好处，我们开发了一个轻量级服务器。只有方法GET和POST以及定义硬截止时间的头文件可用。 RTEWS在C ++中在Altera Cyclone II FPGA内置的Nios II处理器上运行的MicroC / OS-II环境中实现。接下来，我们进行了显示请求处理的有效性的实验。服务器已经测试，每秒增加的传入请求数。该测试在两遍中进行。首先，所有请求都按FIFO顺序进行，就像现有的Web服务器一样。在第二遍，我们的调度策略被应用。并发请求的数量已从10个增加到3000个。对于每个情况，分析和计数取消的实时请求的数量。结果如图5所示.X轴显示传入请求的数量，Y轴显示服务器取消的RT请求的百分比，因为它们不符合截止日期。服务器执行了几个不同的任务作为请求的处理程序。这些任务的平均执行时间在100到3000 m的范围内。RT请求的期限不高于3500 ms。

可以观察到，在HTTP服务器中获得的丢失的实时请求的数量是在RTEWS中的几倍。即使对于大量的并发请求，RTEWS中丢失的RT请求的数量也不超过5％。对于HTTP服务器，RT请求在40％的此类请求中不满足时间要求。对于大量的并发请求，此百分比增加到80-90％。

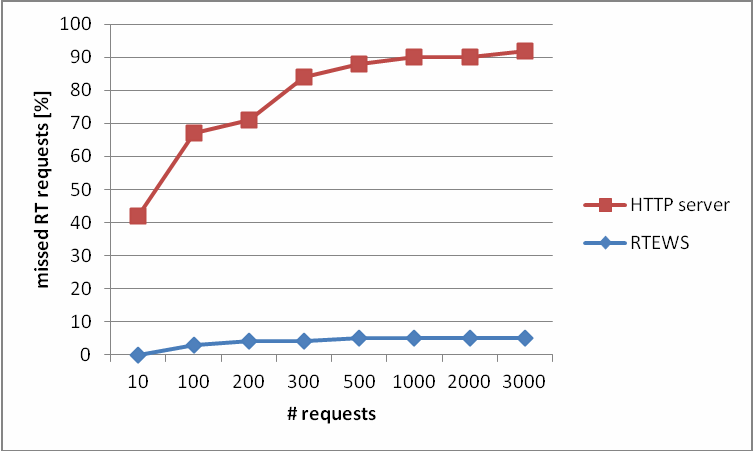


图. 5. .错过的实时请求数。

* 1. 结论

在本文中提出了嵌入式实时Web服务器的架构。服务器考虑硬和软的期限以及预期的服务器负载来调度HTTP请求。通过使用HTTP协议的建议扩展，可以实现最终期限的规范。请求处理程序的LLF调度保证找到适当的调度（如果存在）和高质量的服务。如果不能满足截止时间，服务器取消请求并向客户端发送适当的响应。

实验结果表明，在所提出的服务器中错过的实时请求的数量比使用现有的HTTP服务器小几倍。由于在我们的方法中，不能在预期时间段内提供的请求被取消，因此避免了服务器的过载或下降。

该服务器可以在根据感测即服务商业模型使用嵌入式系统的IoT应用中非常有用。我们相信，在不久的将来，越来越多的配备不同传感器并位于不同地方的嵌入式系统将支持不同Web应用程序的实时服务。

REFERENCES

[1] Atzoria, L., Iera, A., Morabito, G.: The Internet of Things: A survey. Computer Networks, vol. 54, no.15, pp. 2787- 2805 (2010).

[2] Höller, V. Tsiatsis, C. Mulligan, S. Karnouskos, S. Avesand, D. Boyle: From Machine-to-Machine to the Internet of Things: Introduction to a New Age of Intelligence. Elsevier, 2014.

[3] Xiang Sheng; Jian Tang; Xuejie Xiao; Guoliang Xue, "Sensing as a Service: Challenges, Solutions and Future Directions‖, *IEEE Sensors Journal,* vol.13, no.10, pp.3733-3741, Oct. 2013.

[4] Dargie, W. and Poellabauer, C., "Fundamentals of wireless sensor networks: theory and practice", John Wiley and Sons, 2010.

[5] R. Fielding, J. C. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee, ‖Hypertext Transfer Protocol - HTTP/1.1‖, IETF, RFC 2616, 1999.

[6] Guinard, D., Trifa, V., Mattern, F., Wilde, E.: From the Internet of Things to the Web of Things: Resource Oriented Architecture and Best Practices. In: Uckelmann, Dieter; Harrison, Mark; Michahelles, Florian (Eds.) Architecting the Internet of Things, Springer-Verlag Berlin Heidelberg (2011).

[7] H. Schulzrinne, A. Rao, R. Lanphier, *Real Time Streaming Protocol (RTSP)*, IETF, RFC 2326,1998

[8] Peter Saint-Andre, Kevin Smith, Remko TronconPeter Saint-Andre, Kevin Smith, Remko Troncon, XMPP: The Definitive Guide, [O'Reilly Media,](http://it-ebooks.info/publisher/3/) 2009.

[9] Crane, Dave; McCarthy, Phil: Comet and Reverse Ajax: The Next-Generation Ajax 2.0. Apress, (2008).

[10] F. M. Aymerich, G. Fenu, S. Surcis, ―A real time financial system based on grid and cloud computing‖ *ACM symposium on Applied Computing*, March 2009, New York, pp 1219–1220.

[11] S. Liu, G. Quan, S. Ren, ―On-Line Scheduling of Real- Time Services for Cloud Computing‖ *World Congress on Services*, July 2010, Miami, pp 459–464.

[12] W. Tsai, Q. Shao, X. Sun, J. Elston, ―Real-Time Service- Oriented Cloud Computing‖ *World Congress on Services*, July 2010, Miami, pp 473–478.

[13] K. H. Kim, A. Beloglazov, R. Buyya, ―Power-aware provisioning of cloud resources for realtime services‖ *International Workshop on Middleware for Grids, Clouds and e-Science*, 2009, New York.

[14] K. Kumar, J. Feng, Y. Nimmagadda, Y. Lu, ―Resource Allocation for Real-Time Tasks using Cloud Computing‖ *International Conference on Computer Communications and Networks (ICCCN)*, July 2011, pp. 1-7.

[15] S. Bąk, R.Czarnecki, S. Deniziak, "Synthesis of Real- Time Applications for Internet of Things", LNCS vol.

7719, pp. 37-51, 2013.

[16] S. Bąk, R. Czarnecki, S. Deniziak "Synthesis of real-time cloud applications for Internet of things" *Turkish Journal of Electrical Engineering and Computer Sciences*, to be published, [http://dx.doi.org/10.3906/elk-1302-178.](http://dx.doi.org/10.3906/elk-1302-178)

[17] Deniziak, S.; Bak, S., "Synthesis of real time distributed applications for cloud computing", Proc of the IEEE *Federated Conference on Computer Science and Information Systems (FedCSIS),* pp.743-752, 2014.

[18] Embedthis Software, Appweb Architecture, [http://appwebserver.org/products/appweb/doc/ref/architect](http://appwebserver.org/products/appweb/doc/ref/architecture.html) [ure.html,](http://appwebserver.org/products/appweb/doc/ref/architecture.html) 2014.

[19] Unicoi Systems, Fusion Embedded™ HTTP, [http://www.unicoi.com/product\_briefs/http.pdf,](http://www.unicoi.com/product_briefs/http.pdf) 2013.

[20] Real Time Logic, Barracuda Web Server, [https://realtimelogic.com/products/barracuda-web-server/,](https://realtimelogic.com/products/barracuda-web-server/) 2014.

[21] Duquennoy, S.; Grimaud, G.; Vandewalle, J.-J., "Smews: Smart and Mobile Embedded Web Server", *International Conference on Complex, Intelligent and Software Intensive Systems, 2009. CISIS '09,* pp.571-576, 2009.

[22] Trotter Cashion, Introducing Chloe - The Realtime Web Server, [http://www.trottercashion.com/2011/06/13/intro-](http://www.trottercashion.com/2011/06/13/intro-ducing-chloe.html) [ducing-chloe.html,](http://www.trottercashion.com/2011/06/13/intro-ducing-chloe.html) 2011.

[23] S. Loreto, M. Thomson, G. Wilkins, ―Hypertext Transfer Protocol (HTTP) Timeouts‖, IETF, draft-loreto-http- timeout-00, 2010.

[24] G. C. Buttazzo *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*, Kluwer Academic Publishers, 1997.

**作者简介**



**Radosław Czarnecki**  是波兰克拉科夫理工大学电气与计算机工程学院计算机科学系的助理教授。 2002年，他获得克拉科夫理工大学电气工程硕士学位，以及2008年ZielonaGóra大学物理互联网时代应用计算机科学博士学位。

他的研究的主要侧重于设计实时物联网系统的方法。 先前的研究针对硬件 - 软件分布式系统和可重配置嵌入式系统的综合方法。 他定期在期刊和会议论文中发表了嵌入式系统的科学论文和这些系统的综合方法，并在国家和国际会议上介绍了这些论文，以促进研究。 他是IEEE期刊上关于物联网实时应用综合的两篇论文的作者。

**Stanisław Deniziak** 是波兰基尔斯科技大学计算机科学系计算机科学教授。 他在华沙理工大学获得计算机科学硕士学位，在格但斯克理工大学获得博士学位。 2006年，他获得华沙理工大学计算机科学硕士学位。 现在，他是凯尔采工业大学电气工程，自动化和计算机科学学院的研究和推广副院长。

他在各种国际和国家期刊和会议上发表了78篇研究论文。 他是系统和软件，计算，微处理器和微系统杂志，国际应用数学和计算机科学杂志，开放控制论与系统杂志，国际物理科学杂志，年鉴UMCS的7个国际期刊的活跃审稿人结束编辑成员， 教学A Informatica。 他回顾了许多国际会议的研究论文，如：IEEE设计自动化大会，科学与工程计算方法国际会议等。

Deniziak教授是IEEE和IEEE计算机学会会员。

**How to cite this paper:** Radosław Czarnecki, Stanislaw Deniziak,"Embedded Real-Time HTTP Server", IJCNIS, vol.7, no.5, pp.1-8, 2015.DOI: 10.5815/ijcnis.2015.05.01

**Open Source Based Privacy-Proxy to Restrain Connectivity of Mobile Apps**

Alexander Mense University of Applied Sciences Technikum Wien

Vienna, Austria [mense@technikum-wien.at](mailto:mense@technikum-wien.at)

Sabrina Steger

University of Applied Sciences Technikum Wien Vienna, Austria

[ic14m010@technikum-wien.at](mailto:ic14m010@technikum-wien.at)

Dragan Jukic-Sunaric University of Applied Sciences Technikum Wien

Vienna, Austria [ic14m035@technikum-wien.at](mailto:ic14m035@technikum-wien.at)

András Mészáros University of Applied Sciences Technikum Wien

Vienna, Austria [ic14m042@technikum-wien.at](mailto:ic14m042@technikum-wien.at)

Matthias Sulek

University of Applied Sciences Technikum Wien Vienna, Austria

[ic14m033@technikum-wien.at](mailto:ic14m033@technikum-wien.at)

**ABSTRACT**

Mobile Devices are part of our lives and we store a lot of private information on it as well as use services that handle sensitive information (e.g. mobile health apps). Whenever users install an application on their smartphones they have to decide whether to trust the applications and share private and sensitive data with at least the developer-owned services. But almost all modern apps not only transmit data to the developer owned servers but also send information to advertising-, analyzing and tracking partners. This paper presents an approach for a “privacy- proxy” which enables to filter unwanted data traffic to third party services without installing additional applications on the smartphone. It is based on a firewall using a black list of tracking- and analyzing networks which is automatically updated on a daily basis. The proof of concept has been implemented with open source components on a Raspberry Pi.

Categories and Subject Descriptors

ACM 1998: **• K.4.1 Computers and Society:** Public Policy Issues – *Privacy;* **• C.2.0 Computer-Communications Networks:** General **–** *Security and Protection*

ACM 2012: **• Security and privacy**: Security services – *Pseudonymity, anonymity and untraceability; Privacy-preserving protocols.*

General Terms

Security

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from [Permissions@acm.org.](mailto:Permissions@acm.org)

MoMM '16, November 28-30, 2016, Singapore, Singapore

© 2016 ACM. ISBN 978-1-4503-4806-5/16/11…$15.00

DOI: <http://dx.doi.org/10.1145/3007120.3007163>

Keywords

Mobile Apps; Security; Privacy; Sensitive Information; Proxy.

1. INTRODUCTION

With their big variety of functions and their connectivity mobile devices have partially replaced the Personal Computer. They enable access to email, provide calendars to save all your appointments, ensure connectivity with your friends and because of the thousands of apps their functionalities are endless extendable. But they also store large amount of personal data as well as information about others like friends or business partners. There are more benefits: they are able to supply sensors and with the help of these sensors it is possible to use GPS for navigation, to use Bluetooth to collect information about your heart rate or to make pictures wherever you are. All these small parts together make mobile devices currently best opportunity available for getting private data as well as profiling, e.g. GPS data can give hints about the workplace and also the living address. Collected health information makes it furthermore possible to gain knowledge about the physical health [1]. While third-party applications open even more possibilities for smartphones, through access to user data they serve as a tool for profiling. Therefore it seems it is not possible to stay anonymous anymore [2].

It has been shown in several different evaluations [1] [2], [3], [4] that a big part of the mobile apps have security and/or privacy issues. A study from Huckvale et al [4] about the analysis of apps included in NHS England's Health Apps Library shows the lack of security and privacy of many mHealth apps and reports that most apps do not handle data according to their privacy policy, and that some apps do not even have any privacy policy.

One of the biggest problems is that almost no mobile app only connects to the developer-owned backend services, but also contacts third-party websites for advertising, analytics ad tracking. The communication with the advertising sites occurs mainly unencrypted. Some of the analyzed applications also send “usage data” in plain text to third-party advertisers after e.g. measuring fitness activities in a fitness app.

1. GOAL AND CONCEPT

There are already software products for mobile devices, which offer functionalities to limit connectivity (e.g. firewalls), limit access to data (SRT AppGuard) or directly try to enhance privacy (e.g. TaintDroid). But they have to be installed directly on the device and usually need a rooted device.

In order to hinder sniffing activities of apps we choose a different approach, which is able to prevent that advertising-, tracking- and analyzing networks are contacted by the apps.

The concept is to provide a service in the network (proxy), forward all network traffic from the mobile device to the where the traffic is analyzed. In a first step the goal for the work described in this paper was to disable connections to ad-, analytics- and tracking services. In a further step network traffic can be analyzed to identify and prevent possible leakage of sensitive data for a better protection of privacy.

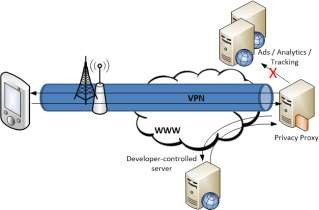
To block connections to unwanted service the following components are employed:

* + Use a VPN to tunnel all network traffic from the mobile device to the proxy service
  + Use a firewall to filter and block traffic to ads-, analytics- and tracking services
  + Automatically update firewall rules based on a list of ads-, analytics- and tracking services

Figure 1 shows the usual data flow from mobile apps, figure 2 illustrates the concept of the privacy proxy.



**Figure 1: Typical mobile app connections**



**Figure 2: Connections using the privacy proxy concept**

1. **IMPLEMENTION**

For a proof of concept the service was implemented on a Rasperry Pi using open source tools and components.

This section secure communication from the mobile device to the Raspberry Pi is described and the second part explained the filtering based on IPTables firewall blacklists and shows an automatic solution to update the unwanted domains on a daily basis.

* 1. Hardware - Raspberry Pi

The use of Raspberry Pi is a cheap, always-on, and simple way for users to create their own VPN endpoint. Raspberry Pi is based on open source software and therefore it is a transparent and secure solution for protecting privacy.

* 1. VPN - strongSwan

To be sure that every single packet is being sent to the Raspberry the VPN connection needs to be stable over long periods of time. Also the different smartphone operating systems need to offer the same IPSec possibilities. One possibility which is actually quite easy to implement with Raspberry Pi packages and is also supported by iOS and Android devices is IPSec with the Extensible Authentication Protocol, in short IPSec XAUTH.

The standard IPSec packages for the Raspberry Pi are openSwan1 (now named libreSwan) and strongSwan2 (strongSwan is available through non-standard repositories). But because of the higher stability, the more frequent update cycles, and the better documentation we decided to use strongSwan [Z] instead of openSwan. For the XAUTH compatibility the charon library and some extra packages (libcharon-extra-plugins) need to be installed. IKEv1 was used to be compatible also older versions of mobile operation systems.

* + 1. Authentication

For the authentication the XAUTH option with pre-shared keys is used. Afterwards strongSwan should look for a username/password combination in the ipsec.conf file. As this is a very simple installation, the username and the password are stored in cleartext in the ipsec.conf file. For the routing/forwarding options the sysctl.conf has to be modified, as it needs to forward packets. To provide stronger security different RSA methods are provided by strongSwan. For future implementations we recommend using at least a server certificate authentication prohibiting man-in-the-middle attacks.

* + 1. Stability

As the Raspberry Pi has to be reachable at any time we propose the use of a dynamic DNS solution for the private home network, in case the internet service provider (ISP) is changing IP addresses frequently. We also recommend editing the Raspberry Pi’s interfaces file and assign a static IP address. Unfortunately the Raspberry Pi WiFi driver is poorly implemented, resulting in low VPN speed and/or connection losses. Thus we suggest using cable over WiFi. In order to enable VPN connection the ISP modem needs to be able to do port-forwarding on UDP port 500 and 4500 from the official IP to the Raspberry Pi private/local IP.

* + 1. Security

In association with strongSwan [7] a large variety of IKEv1 cipher suites are being offered [8]. This is a clear advantage, as the offered cipher suites on the smartphones vary from operating system to operating system and also from version to version. The selected suite for a specific connection is visible in

/var/log/daemon.log. For example, below is the IPSec stack, primarily chosen by android devices:

IKE:AES\_CBC\_256/HMAC\_SHA2\_256\_128/PRF\_HMAC\_SH A2\_256/MODP\_1024

The first part describes the encryption, the second and third the integrity hashing algorithms used and the final part the Diffie Hellman groups used for the key exchange process. The example above shows that the used protocols offer a high grade of security for private VPNs. With strongSwan we ensure that the established tunnel, although open source software is being used, offer a high level of security.

* + 1. Testing

By keeping the tunnel alive over a longer period of time (about two hours) the stability was tested. If the device reaches a Telecom dead zone, the re-establishing process occurred immediately and no further problems were detected. To minimize traffic being sent over the mobile data network, we recommend compressing the data stream. strongSwan, per default does not compress data streams. However this feature can be easily activated with the compress=yes parameter in the ipsec.conf file. Unfortunately this feature can cause incompatibility with some iOS and Android versions.

* 1. Traffic control with IPTables

Blocking of unwanted connections to ad-, analytics- or tracking- services is done with the IPTables firewall [X] which is part of almost all Unix/Linux operating system versions.

* + 1. Automatic update of hosts to block

A static blocking list which needs to be manually updated periodically is not an optimal and permanent solution. Therefore the list of hosts to block needs to be adapted automatically on a daily basis. The privacy service has to cope with this ever changing environment. The automatic update is done on the basis of a black list containing ad-, analytics- and tracking servers.

* + 1. Blacklist provider

There are several providers who publish lists of tracking and advertising hosts. To select a blacklist we defined two criteria: the reliability/reputation of the blacklist provider and regular updates of the list. Based on these criteria Shalla Secure Service KG [9] was selected which provides a blacklist with over 1.7 million entries. Additionally it is free of charge for personal usage.

Blacklist format

The used lists contain domain names and IP addresses which need to be transformed for the use with IPTables, as it is primarily designed to work with IP addresses. Furthermore periodical control of the domain names for changed IP’s is needed.

* + 1. Requirements and Tools

For the automatic update of the filter tables for IPTables we defined the following requirements:

* + - * Automatic run, without user action
      * Possible use with different list providers
      * Easy deployment and use on the Raspberry Pi
      * The ability to download the source list from the internet, but also read local definitions
      * Fast update of the blocking rules in IPTables (to minimize time of only partial protection)

To achieve these goals and implement the functionalities a combination of the Perl scripting language and the Linux Bash Shell scripting was used. Both are available over Raspberry PI’s repository as well. Among the scripting languages Perl is the best for text processing. The language design includes built-in functions for regular expressions, which makes the manipulation of text-based files easy. For file and directory management, we decided to use callout to Bash Shell scripts, which provides a more compact and clear method. To achieve an automatic start of the “HostlistLoader” tool the built-in Linux Cron is used.

* + 1. “HostlistLoader”

The main parts of the developed application to load the list with the hosts to block and to convert them into appropriate firewall rules are:

* + - * HostlistLoaderConfig.xml (The configuration file) defines parsers, which interpret the sources.
      * LocalFileParser: The LocalFileParser module loads only the locally stored list of blacklisted hosts (defined either by IP or by domain name).
      * InternetParser: The InternetParser module loads a TAR GZIP file from the Shalla Secure Services KG5 provider. This parser contains some special actions, like preparing the loaded file. In the current case the preparation is implemented in a different Shell script, the defaultInternetParser\_prepare.sh file, which extracts the source compressed file in a temporary directory, selects the tracking and advertising lists, concatenates them and places the resulting file to a defined directory. Further processing of the prepared list happens subsequently in the parser module.
      * StartupFileParser: The StartupFileParser module is a built-in optimization for a special case: the system startup. At system start, the resources of the system are heavily loaded with system tasks and a complete cold start would take longer time.

The whole “HostlistLoader” automation tool consists of severeal Perl modules for configuration management, maintaining the IP list in the memory, converting and applying the IP list as IPTables rule sets as well as logging.

* + 1. Optimization

All parser modules (except the special built-in StartupFileParser) contain a domain caching method. If the source, loaded from the providers, contains domain names, the automation resolves them, but it takes more time to do so (approximately from a few hundred milliseconds to a few seconds, depending on the network). As the IP assignment to domain names changes relatively rarely it makes another optimization possible. The resolved domain names and the resulting IP addresses can be stored in a defined domain cache file for all the parser modules defining the cache. As a mandatory attribute (keep\_days), the validity period should be given. The automation always checks the age of the cache, and if needed, it makes a new domain name resolution.

* + 1. Logging

The updating module implements a comprehensive logging. All the successful steps are indicated by an “[INFO]” tag and all problems are marked by “[ERROR]” tags. This makes filtering of the occurring problems easy. The output log file is also configurable through the start parameters of the main script.

* + 1. IPSet

During the tests we experienced slow performance if large amount of IPTables rules were used. Therefore we switched from IPTables to IPSet [10] which is a framework included in the Linux kernel. IPSet performs the same task 11-times faster than IPTables [11]. Furthermore IPSet also automatically can block the whole port range for a specific IP address. With this it is guaranteed that hidden communication over nonstandard ports is also blocked.

After all parser modules have finished their work, the IPSet module takes the list of IP addresses created in the memory and generates an IP hash set in the Linux system. If the set already exists, it erases the old set and fills in the newly constructed list. At this point an approximately two to five minutes long “non- protection” gap occurs. Future improvement might be to change the flushing mechanism to a synchronising mechanism, closing this small security gap.

1. DISCUSSION AND FURTHER WORK

The concept of implementing privacy controls in form of intermediary services is not fundamentally new. There has been work in the area of protection of user location data (e.g. [12]) as well as rule-based release of data (e.g. [13]). Many models propose to upload data to trusted stores and control access of application to this data, which is quite different to the concept of real-time filtering of data streams. Davies et al. did similar work in the area of Internet of Things (IoT) [14]. They proposed an architecture connecting IoT devices with specific data drivers to “Privacy Mediators” running in private Cloudlets [14]. To some extent our privacy proxy can be interpreted as such a mediator service.

But in difference in the presented approach there is no need for specific data connectors as all network traffic is routed through the VPN to the filtering service. Such a network proxy is definitely a straight forward way to control connections and data traffic of mobile apps.

Data traffic of mobile applications can be easily analyzed by and - as it is shown in section 4 - it is easy to implement. But such a proxy could also be an effective service to scan and analyze the network traffic to detect data leakage.

The biggest advantage of the proxy solution is the displacement of the logic from the mobile device to a server architecture. No app installation, no rooting and/or flashing of the device is needed. The user is able to use the mobile device operating system’s native VPN functions to establish a secure connection to the user’s own Raspberry Pi. This way of transferring the data from a privately controlled device guarantees that no third party can steal them. The Raspberry Pi uses a freely available and updated blacklist to make blocking decisions. The mobile device only has to support VPNs. Various other infrastructure solutions would require more logic on the client side and would therefore not lead to a universal solution.

The traffic from the servers can be easily controlled with a firewall and furthermore the data can be classified and analyzed with a variety of methods such as pattern matching (using regular expressions or word blacklisting), taint analysis or machine learning.

Filtering the network traffic with a firewall is quite normal in any enterprise environment and usually does not have any great

performance impacts on outgoing traffic. As the current solution of the proxy uses an optimized firewall implementation there are only minimal performance implications.

The proof-of-concept implementation is based on a Rasperry Pi and enables private configuration and usage of the proxy. But the concept can be also implemented in a cloud based environment as “privacy-as-a-service”.

1. REFERENCES

[1] M. B. Barcena, C. Wuesst, H. Lau, “How safe is your quantified self?”, Symantec, August 2014. [https://www.symantec.com/content/dam/symantec/docs/whit](https://www.symantec.com/content/dam/symantec/docs/white-papers/how-safe-is-your-quantified-self-en.pdf) [e-papers/how-safe-is-your-quantified-self-en.pdf,](https://www.symantec.com/content/dam/symantec/docs/white-papers/how-safe-is-your-quantified-self-en.pdf) Last access: 20.08.2016

[2] A. Barczok, S. Porteck, “Schutz vor Schnüffel-Apps,” Appgehört, Schnüffel-Apps einschränken, April 2015.

[3] A. MENSE, A. Steger, M. Sulek, D. Jukic-Sunaric, A. Mészáros: Analyzing privacy risks of mHealth applications. In: Transforming Healthcare with the Internet of Things, Studies in Health Technology and Informatics, vol 221, IOS Press, 2016, ISBN 978-1-61499-632-3 (print) | 978-1-61499-

633-0 (online), doi:10.3233/978-1-61499-633-0-41

[4] K. Huckvale, J. Tomás Prieto, M. Tilney, P.-J. Benghozi, and

J. Car, Unaddressed privacy risks in accredited health and wellness apps: a cross-sectional systematic assessment. BMC Medicine 2015, 13:214. doi:10.1186/s12916-015-0444-y

[5] Wu-Chen Su, A Preliminary Survey of Knowledge Discovery on Smartphone Applications (apps): Principles, Techniques and Research Directions for E-health, International Conference on Complex Medical Engineering, pp.369-374, IEEE Computer Society Press, June 2014

[6] <https://www.openswan.org/>

[7] <https://www.strongswan.org/>

[8] StrongSwan: IKEv1 Cipher Suites. [https://wiki.strongswan.org/projects/1/wiki/IKEv1CipherSuit](https://wiki.strongswan.org/projects/1/wiki/IKEv1CipherSuites) [es,](https://wiki.strongswan.org/projects/1/wiki/IKEv1CipherSuites) Last access: 20.08.2016

[9] Shalla Secure Services KG: Shalla Secure Services - Shalla's Blacklists. [http://www.shallalist.de](http://www.shallalist.de/)

[10] <http://ipset.netfilter.org/>

[11] d(a)emonkeeper's purgatory: Mass-blocking IP addresses with ipset. [http://daemonkeeper.net/781/mass-blocking-ip-addresses-with-ipset,](http://daemonkeeper.net/781/mass-blocking-ip-addresses-with-ipset)

Last access: 20.08.2016

[12] A. R. Beresford, F. Stajano: Location privacy in pervasive computing. IEEE Pervasive Computing, 2(1), pp46-55, 2003.

[13] M. Mun, S. Hao, N. Mishra, K. Shilton, J. Burke, D. Estrin,

M. Hansen, R. Govindan: Personal data vaults: a locus of control for personal data streams. In Proc. of ACM CoNEXT, p17, 2010.

[14] N. Davies, N. Taft, M. Satyanarayanan, S. Clinch, B. Amos: Privacy Mediators: Helping IoT Cross the Chasm. In Proceeding HotMobile '16 Proceedings of the 17th International Workshop on Mobile Computing Systems and Applications, pp 39-44, ACM New York, NY, USA 2016, ISBN: 978-1-4503-4145-5, doi: 10.1145/2873587.2873600

**基于开源的隐私代理约束**

**移动应用的连接**

Alexander Mense University of Applied Sciences Technikum Wien

Vienna, Austria [mense@technikum-wien.at](mailto:mense@technikum-wien.at)

Sabrina Steger

University of Applied Sciences Technikum Wien Vienna, Austria

[ic14m010@technikum-wien.at](mailto:ic14m010@technikum-wien.at)

Dragan Jukic-Sunaric University of Applied Sciences Technikum Wien

Vienna, Austria [ic14m035@technikum-wien.at](mailto:ic14m035@technikum-wien.at)

András Mészáros University of Applied Sciences Technikum Wien

Vienna, Austria [ic14m042@technikum-wien.at](mailto:ic14m042@technikum-wien.at)

Matthias Sulek

University of Applied Sciences Technikum Wien Vienna, Austria

[ic14m033@technikum-wien.at](mailto:ic14m033@technikum-wien.at)

**摘要**

移动设备是我们生活的一部分，我们存储大量的私人信息，以及使用处理敏感信息的服务（例如移动健康应用程序）。每当用户在智能手机上安装应用程序时，他们必须决定是否信任应用程序，并至少与开发人员拥有的服务共享私人和敏感数据。但是几乎所有的现代应用程序不仅将数据传输到开发人员拥有的服务器，而且还向广告，分析和跟踪合作伙伴发送信息。本文提出了一种“隐私代理”方法，可以过滤不需要的数据流量到第三方服务而无需在智能手机上安装其他应用程序。它基于防火墙，使用黑名单跟踪和分析网络，这是每天自动更新。概念验证已经在Raspberry Pi上使用开源组件实现。

类别和主题描述符

ACM 1998: **• K.4.1 计算机和社会:** 公共政策问题 – *隐私;* **• C.2.0 计算机通信网络:** 一般条款**–** *安全和保护*

ACM 2012: **• 安全和隐私**: 安全服务 – *假名、 匿名性和不可追踪性;隐私保护的协议。*

通用条款

安全

使数字或所有的硬拷贝或个人或教室使用此工作的一部分便可获得免费副本不是制作或发行利润或商业优势和副本第一页上承担本通知和充分的引文。版权的组件的其他人拥有的不是 ACM 必须被尊重这项工作。抽象与信用被允许。否则，复制或重新发布，在服务器上张贴或重新分配到列表，需要事先特定权限和/或费用。从 Permissions@acm.org 请求的权限。© 2016 ACM MoMM 16，2016 年 11 月 28-30 日，新加坡，新加坡。© 2016 ACM. ISBN 978-1-4503-4806-5/16/11…$15.00

DOI: <http://dx.doi.org/10.1145/3007120.3007163>

**关键词**

移动应用; 安全; 隐私; 敏感信息; 代理。

1. 介绍

由于其各种各样的功能和连接性，移动设备部分地取代了个人计算机。他们允许访问电子邮件，提供日历保存所有的约会，确保与朋友的连接，并且由于成千上万的应用程序，他们的功能是无尽的可扩展。但他们还存储大量的个人数据以及关于他人的信息，如朋友或业务伙伴。还有更多的好处：他们能够提供传感器，并借助这些传感器，可以使用GPS导航，使用蓝牙收集有关你的心率的信息或无论你在哪里做图片。所有这些小部件一起使得移动设备当前是用于获得私人数据以及分析（例如， GPS数据可以提供关于工作场所和生活地址的提示。收集的健康信息使得进一步有可能获得关于身体健康的知识[1]。虽然第三方应用程序为智能手机打开更多的可能性，通过访问用户数据，它们作为一个工具进行分析。因此似乎不可能保持匿名[2]。

它已经显示在几个不同的评估[1 ] [2]，[3]，[4]大部分的移动应用程序有安全和/或隐私问题。 Huckvale等人[4]对NHS英格兰健康应用程序库中包含的应用程序的分析显示，许多移动健康应用程序缺乏安全和隐私，大多数应用程序根据其隐私政策不处理数据，应用程序甚至没有任何隐私政策。

一个最大的问题是几乎没有移动应用程序只连接到开发人员拥有的后端服务，但也联系第三方网站广告，分析广告跟踪。与广告网站的通信发生主要是未加密的。一些分析的应用程序也发送“使用数据“以纯文本格式发送给第三方广告客户。测量健身应用程序中的健身活动。

1. 目标和理念

已经存在用于移动设备的软件产品，其提供用于限制连接（例如防火墙），限制对数据的访问（SRT AppGuard）或直接尝试增强隐私（例如TaintDroid）的功能。但是它们必须直接安装在设备上，通常需要有根的设备。

为了阻止应用程式的嗅探活动，我们选择不同的方法，这可以防止广告，跟踪和分析网络由应用程序联系。

该概念是在网络（代理）中提供服务，将来自移动设备的所有网络流量转发到分析流量的地方。第一步，本文中描述的工作的目标是禁用与广告，分析和跟踪服务的连接。在进一步的步骤中，可以分析网络流量以识别和防止敏感数据的可能泄漏，以便更好地保护隐私。

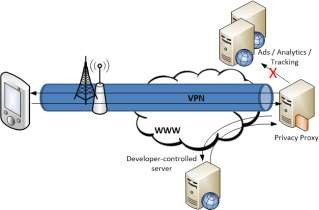
要阻止到不需要的服务的连接，使用以下组件：

* + 使用VPN将所有网络设备隧道传送到代理服务
  + 使用防火墙过滤和拦截来自广告，分析和跟踪服务的流量
  + 自动更新基于广告，分析和跟踪服务的防火墙规则

图1显示了来自移动应用的常见数据流,图2说明了隐私代理的概念。



**图1: 典型的移动应用连接**



**图2: 使用隐私代理概念的连接**

1. **实施**

为了证明概念，服务是使用开源工具和组件在Rasperry Pi上实现的。

本节描述了从移动设备到Raspberry Pi的安全通信，第二部分解释了基于IPTables防火墙黑名单的过滤，并显示了每天更新不需要的域的自动解决方案。

* 1. 硬件-树莓派

使用Raspberry Pi是一种便宜，永远在线，简单的方式，用户可以创建自己的VPN端点。 Raspberry Pi基于开源软件，因此它是一个用于保护隐私的透明，安全的解决方案。

* 1. VPN - strongSwan

为了确保每个单个数据包都被发送到Raspberry，VPN连接需要长时间稳定。不同的智能手机操作系统需要提供相同的IPSec可能性。实际上很容易用Raspberry Pi软件包实现的一个可能性，也是iOS和Android设备支持的一种可能性是IPSec与可扩展身份验证协议，简称IPSec XAUTH。

Raspberry Pi的标准IPSec包是openSwan1（现在命名为libreSwan）和strongSwan2（strongSwan可通过非标准存储库使用）。但是由于更高的稳定性，更频繁的更新周期，以及更好的文档，我们决定使用strongSwan [Z]而不是openSwan。对于XAUTH兼容性，需要安装charon库和一些额外的包（libcharon-extra-plugins）。 IKEv1用于兼容旧版本的移动操作系统。

* + 1. 认证

对于认证，使用具有预共享密钥的XAUTH选项。之后，strongSwan应该在ipsec.conf文件中查找用户名/密码组合。由于这是一个非常简单的安装，用户名和密码以ipsec.conf文件中的明文存储。对于路由/转发选项，sysctl.conf必须修改，因为它需要转发数据包。为了提供更强的安全性，strongSwan提供了不同的RSA方法。对于将来的实施，我们建议至少使用一个禁止中间人攻击的服务器证书身份验证。

* + 1. 稳定性

由于Raspberry Pi必须随时可以访问，所以我们建议为私人家庭网络使用动态DNS解决方案，以防互联网服务提供商（ISP）频繁更改IP地址。我们还建议编辑Raspberry Pi的interfaces文件并分配静态IP地址。不幸的是，Raspberry Pi WiFi驱动程序实施不良，导致低的VPN速度和/或连接损失。因此，我们建议使用电缆通过WiFi。为了启用VPN连接，ISP调制解调器需要能够在UDP端口500上进行端口转发

和4500从官方IP到Raspberry Pi私有/本地IP。

* + 1. 安全

与strongSwan [7]相关联，提供了大量IKEv1密码套件[8]。这是一个明显的优势，因为在智能手机上提供的密码套件因操作系统和操作系统而异，并且从版本到版本也不同。用于特定连接的所选套件在/var/log/daemon.log中可见。例如，下面是IPSec堆栈，主要由android设备选择：

IKE:AES\_CBC\_256/HMAC\_SHA2\_256\_128/PRF\_HMAC\_SH A2\_256/MODP\_1024

第一部分描述加密，第二和第三是使用的完整性散列算法，最后一部分是用于密钥交换过程的Diffie Hellman组。上面的示例显示，所使用的协议为私有VPN提供高级别的安全性。使用strongSwan，我们确保建立的隧道，虽然开放源代码软件正在使用，提供高水平的安全性。

* + 1. 测试

通过保持隧道在更长的时间（大约两个小时）内活动，测试了稳定性。如果设备到达电信死区，则重新建立过程立即发生，并且没有检测到进一步的问题。为了最小化通过移动数据网络发送的流量，我们建议压缩数据流。 strongSwan，默认情况下不压缩数据流。但是，可以使用ipsec.conf文件中的compress = yes参数轻松激活此功能。很抱歉，此功能可能会导致与某些iOS和Android版本不兼容。

* + 1. 使用IPTables的流量控制

使用IPTables防火墙[X]来阻止对广告，分析或跟踪服务的不必要的连接，这是几乎所有Unix / Linux操作系统版本的一部分。

* + 1. 自动更新要阻止的主机

需要定期手动更新的静态阻止列表不是最佳和永久的解决方案。因此，要阻止的主机列表需要每天自动调整。隐私服务必须应付这种不断变化的环境。自动更新是在包含广告，分析和跟踪服务器的黑名单的基础上完成的。

* + 1. 黑名单提供程序

有几个提供商发布跟踪和广告主机列表。要选择黑名单，我们定义了两个标准：黑名单提供商的可靠性/信誉和列表的定期更新。基于这些标准，选择了Shalla Secure Service KG [9]，其提供了具有超过170万条目的黑名单。此外，它是免费的个人使用。

* + 1. 黑名单格式

所使用的列表包含需要被转换以与IPTables一起使用的域名和IP地址，因为它主要被设计为与IP地址一起使用。此外，需要对改变的IP的域名进行定期控制。

* + 1. 要求和工具

为了自动更新IPTables的过滤器表，我们定义了以下要求：

* + - * 自动运行，无需用户操作
      * 可能用于不同的列表提供程序
      * 在Raspberry Pi上轻松部署和使用
      * 从互联网下载源列表的能力，但也读取本地定义
      * 快速更新IPTables中的阻止规则（以最小化仅部分保护的时间）

为了实现这些目标并实现功能，使用了Perl脚本语言和Linux Bash Shell脚本的组合。两者都可以通过Raspberry PI的存储库。在脚本语言中，Perl是最适合文本处理的。语言设计包括正则表达式的内置函数，这使得基于文本的文件的操作变得容易。对于文件和目录管理，我们决定使用callout to Bash Shell脚本，这提供了一个更紧凑和更清晰的方法。要实现“HostlistLoader”工具的自动启动，使用内置的Linux Cron。

* + 1. “HostlistLoader”

开发的应用程序的主要部分加载列表与要阻止的主机，并将其转换为适当的防火墙规则是：

* + - * HostlistLoaderConfig.xml（配置文件）定义解析器，解释源。
      * LocalFileParser：LocalFileParser模块只加载本地存储的黑名单主机列表（由IP或域名定义）。
      * InternetParser：InternetParser模块从Shalla Secure Services KG5提供程序加载TAR GZIP文件。此解析器包含一些特殊操作，如准备加载的文件。在当前情况下，准备在不同的Shell脚本中实现，defaultInternetParser\_prepare.sh文件在临时目录中提取源压缩文件，选择跟踪和广告列表，连接它们并将生成的文件放置到定义的目录。随后在解析器模块中进行所准备的列表的进一步处理。
      * StartupFileParser：StartupFileParser模块是针对特殊情况的内置优化：系统启动。在系统启动时，系统的资源是严重负载系统任务和完全冷启动将需要更长的时间。

整个“HostlistLoader”自动化工具包括用于配置管理的严格的Perl模块，维护内存中的IP列表，将IP列表转换和应用为IPTablesrule集以及记录。

* + 1. 优化

所有解析器模块（特殊内置的StartupFileParser除外）都包含一个域缓存方法。如果从提供程序加载的源包含域名，则自动解析它们，但这需要更多时间（大约从几百毫秒到几秒，取决于网络）。由于对域名的IP分配变化相对较少，它使得另一个优化成为可能。解析的域名和结果IP地址可以存储在定义缓存的所有解析器模块的定义域缓存文件中。作为强制属性（keep\_days），应当给出有效期。自动化始终检查缓存的时间，如果需要，它进行新的域名解析。

* + 1. 记录

更新模块实现全面的日志记录。所有成功的步骤由“[INFO]”标签指示，所有问题由“[ERROR]”标签标记。这使得对出现的问题的过滤变得容易。输出日志文件也可以通过主脚本的启动参数进行配置。

* + 1. IPSet

在测试过程中，如果使用了大量的IPTables规则，我们的性能就会降低。因此，我们从IPTables切换到IPSet [10]，这是一个包含在Linux内核中的框架。 IPSet执行相同的任务比IPTables的11倍快[11]。此外，IPSet还可以自动阻止特定IP地址的整个端口范围。有了它，保证非标准端口上的隐藏通信也被阻止。

所有解析器模块完成工作后，IPSet模块获取在内存中创建的IP地址列表，并在Linux系统中生成IP哈希集。如果集合已经存在，它将擦除旧集合并填充新构造的列表。在这一点上，出现大约两到五分钟长的“非保护”间隙。未来的改进可能是将冲洗机制改为同步机制，从而弥补这一小的安全漏洞。

1. 讨论和进一步工作

以中介服务的形式实施隐私控制的概念不是根本上新的概念。在保护用户位置数据（例如[12]）以及基于规则的数据释放（例如，[13]）的领域中已经进行了工作。许多模型建议将数据上传到可信存储并控制应用对该数据的访问，这与数据流的实时过滤的概念截然不同。 Davies et al。在物联网（IoT）领域做了类似的工作[14]。他们提出了一种将物联网设备与特定数据驱动程序连接到在私有Cloudlet中运行的“隐私中介器”的架构[14]。在某种程度上，我们的隐私代理可以被解释为这样的调解服务。

但是在所提出的方法的差异中，不需要特定的数据连接器，因为所有网络业务通过VPN路由到过滤服务。这样的网络代理绝对是一个直接的方式来控制移动应用程序的连接和数据流量。

移动应用程序的数据流量可以很容易地通过第4部分进行分析，如第4节所示。但是这样的代理也可以是扫描和分析网络流量以检测数据泄漏的有效服务。

代理解决方案的最大优势是逻辑从移动设备移位到服务器体系结构。不需要应用安装，不需要设备的生根和/或闪烁。用户能够使用移动设备操作系统的本地VPN功能来建立到用户自己的树莓派的安全连接。这种从私人控制设备传输数据的方式保证没有第三方可以窃取它们。树莓派使用免费提供和更新黑名单做阻塞决定。移动设备只需要支持VPN。各种其他基础设施解决方案将需要在客户端更多的逻辑，因此不会导致通用解决方案。

来自服务器的流量可以通过防火墙容易地控制，此外，可以使用各种方法来分类和分析数据，例如模式匹配（使用正则表达式或字黑名单），污点分析或机器学习。

在任何企业环境中使用防火墙过滤网络流量是正常的，并且通常对传出流量没有任何显着的性能影响。因为代理的当前解决方案使用优化的防火墙实现，所以只有最小的性能影响。

概念验证实现基于Rasperry Pi，并支持私有配置和代理的使用。但是该概念也可以在基于云的环境中实现为“隐私即服务”。

1. REFERENCES

[1] M. B. Barcena, C. Wuesst, H. Lau, “How safe is your quantified self?”, Symantec, August 2014. [https://www.symantec.com/content/dam/symantec/docs/whit](https://www.symantec.com/content/dam/symantec/docs/white-papers/how-safe-is-your-quantified-self-en.pdf) [e-papers/how-safe-is-your-quantified-self-en.pdf,](https://www.symantec.com/content/dam/symantec/docs/white-papers/how-safe-is-your-quantified-self-en.pdf) Last access: 20.08.2016

[2] A. Barczok, S. Porteck, “Schutz vor Schnüffel-Apps,” Appgehört, Schnüffel-Apps einschränken, April 2015.

[3] A. MENSE, A. Steger, M. Sulek, D. Jukic-Sunaric, A. Mészáros: Analyzing privacy risks of mHealth applications. In: Transforming Healthcare with the Internet of Things, Studies in Health Technology and Informatics, vol 221, IOS Press, 2016, ISBN 978-1-61499-632-3 (print) | 978-1-61499-

633-0 (online), doi:10.3233/978-1-61499-633-0-41

[4] K. Huckvale, J. Tomás Prieto, M. Tilney, P.-J. Benghozi, and

J. Car, Unaddressed privacy risks in accredited health and wellness apps: a cross-sectional systematic assessment. BMC Medicine 2015, 13:214. doi:10.1186/s12916-015-0444-y

[5] Wu-Chen Su, A Preliminary Survey of Knowledge Discovery on Smartphone Applications (apps): Principles, Techniques and Research Directions for E-health, International Conference on Complex Medical Engineering, pp.369-374, IEEE Computer Society Press, June 2014

[6] <https://www.openswan.org/>

[7] <https://www.strongswan.org/>

[8] StrongSwan: IKEv1 Cipher Suites. [https://wiki.strongswan.org/projects/1/wiki/IKEv1CipherSuit](https://wiki.strongswan.org/projects/1/wiki/IKEv1CipherSuites) [es,](https://wiki.strongswan.org/projects/1/wiki/IKEv1CipherSuites) Last access: 20.08.2016

[9] Shalla Secure Services KG: Shalla Secure Services - Shalla's Blacklists. [http://www.shallalist.de](http://www.shallalist.de/)

[10] <http://ipset.netfilter.org/>

[11] d(a)emonkeeper's purgatory: Mass-blocking IP addresses with ipset. [http://daemonkeeper.net/781/mass-blocking-ip-addresses-with-ipset,](http://daemonkeeper.net/781/mass-blocking-ip-addresses-with-ipset)

Last access: 20.08.2016

[12] A. R. Beresford, F. Stajano: Location privacy in pervasive computing. IEEE Pervasive Computing, 2(1), pp46-55, 2003.

[13] M. Mun, S. Hao, N. Mishra, K. Shilton, J. Burke, D. Estrin,

M. Hansen, R. Govindan: Personal data vaults: a locus of control for personal data streams. In Proc. of ACM CoNEXT, p17, 2010.

[14] N. Davies, N. Taft, M. Satyanarayanan, S. Clinch, B. Amos: Privacy Mediators: Helping IoT Cross the Chasm. In Proceeding HotMobile '16 Proceedings of the 17th International Workshop on Mobile Computing Systems and Applications, pp 39-44, ACM New York, NY, USA 2016, ISBN: 978-1-4503-4145-5, doi: 10.1145/2873587.28736