Enabling Push-based Mobile Web Services for Today's Cell Phone Provider Networks

Sascha Roth

Department of Computer Science
Hochschule Darmstadt – University of Applied Sciences, Germany
sascha.roth@computer.org

Abstract: Increased processing power and connectivity enable an integration of mobile devices into enterprise applications. The de facto standard for an enterprise integration is Web Service technology. Notifying devices of business events requires establishing an inbound connection to the devices. While establishing connections from mobile devices to enterprise systems is straight forward, the inverse way is restricted as cell phone provider networks are applying Network Address Translation assigning private IP addresses to connected devices.

This paper introduces enabling technologies, which significantly simplify an enterprise integration of mobile devices facing issues like frequently changing IP addresses, Network Address Translation and limited device energy. To achieve that, a scalable Tunnel Proxy is introduced offering location transparency for mobile Web Services. Furthermore, the proposed solution is implemented based on a concrete application scenario. Finally, the implementation is evaluated with respect to energy consumption and scalability.

1 Introduction

Mobile devices became a tremendously relevant business market and mobile Internet connectivity increasingly becomes a commodity for end users. Paradigms such as ubiquitous computing enable the integration of mobile devices with enterprise systems. However, Internet-capable mobile devices are only partial, if at all, integrated with enterprise systems. Although, todays mobile devices bring plenty of on-board multimedia gadgets like camera support, lightweight Open-GL support, and Global Positioning System (GPS) sensors, they lack capabilities when it comes to an enterprise integration. Loosely coupled integration of mobile devices and cell phones in particular require Web Service-enabling in terms of providing mobile Web Services [FC05], especially for notifying users of events within their environment.

Google's Android is an open source operating system for mobile platforms. The first Android-based cell phone is T-Mobile's G1 aka. HTC Dream. Mobile devices, such as the G1, are commonly connected to the Internet through an IPv4-based cell phone provider network with a non-transparent infrastructure. Consequently, the provider network must be treated as a black box Internet Service Provider (ISP), in particular since enterprises or end

users cannot influence the provider network's security policies [Ero08, p. 8]. When trying to integrate the G1 into enterprise information systems, a major issue is that most cell phone providers use Network Address Translation (NAT) to reduce the allocation of precious public IP addresses (cf. [KL07, p. 368]). While cell phone provider networks provide Internet access for mobile end users, it is not allowed to connect to devices behind a NAT router, i.e. the devices are unaccessible for inbound TCP connections from the Internet. Because SOAP is based on HTTP which again is based on TCP, this also has an impact on devices providing Web Services within cell phone provider networks. Hence, systems connected to the Internet cannot initiate inbound connections for sending structured data in a reliable way to mobile devices, which are connected to the Internet via a cell phone provider network.

This paper introduces technologies, which significantly simplify the process of an enterprise integration for mobile devices facing issues like frequently changing IP addresses, NAT and limited device energy. Therefore, a communication platform for push-based mobile Web Services provided within cell phone provider networks is shown solving the aforementioned problem in Section 3, based on Google's Android platform. In particular 3 introduces a so-called Tunnel Architecture offering location transparency, i.e. a static endpoint for each mobile device, to consume mobile Web Services. After introducing a prototypical implementation in Section 4, the solution is evaluated in Section 5. Finally, Section 6 summarises this paper and proposes further applications.

2 Related Work

This section focuses on similar projects demonstrating an upcoming trend for integrating mobile devices into enterprise systems. A light-weight version of the Jetty Web Server, called *i-jetty* [Cod09], is an Android-based HTTP servlet container enabling access to the phones resources via web browsers. However, any approach of porting servlet-based Web Service provider libraries, e.g. Apache Axis, failed. Also, this servlet container is unaccessible via the Internet when connected through a cell phone provider network.

Apple's iPhone OS 3.0 is able to receive a so-called *push notification* [App10]. Similar to the approach introduced in this paper, Apple uses persistent connections established from the phone to be able to push new notifications through the network onto the device. In contrast to Apple's solution, the here shown approach goes beyond a push-based notification of mobile applications. Since for today's Web Services SOAP, formerly known as Simple Object Access Protocol, is the most popular communication protocol [PvdH07], especially when it comes to an enterprise integration, enabling mobile devices for a Web Service-based integration is crucial. As a result, the proposed approach is service-oriented-architecture (SOA)-ready, e.g. for an integration into a workflow engine using Web Service-* standards like BPEL. Moreover, Apple's push notification is a proprietary solution designed for iPhones only, and thus cannot be applied to every platform.

Research in Motion (RIM) provides push messages applying the Push Access Protocol (PAP) [Wir01]. Thus, the initial message to push on a mobile device is encapsulated within

the Short Message System Centre (SMS-C), such that the message appears as it is sent via the ordinary Short Message Service (SMS). I.e. this protocol is based on the SMS standard, and is therefore cost intensive and unreliable. This technical limitation leads to a financial issue when applying at enterprise scale (cf. [Rot09]).

Milagro et. al. [LRF+09] use a similar approach to provide mobile Web Services within a cell phone provider network using a peer-to-peer (P2P) architecture. A mobile Web Service is identified with the Hydra Identifier (HID) carrying context information in order to route through the P2P network. In contrast, the here shown approach uses a centralised service registry identifying mobile Web Services with a service name and an International Mobile Subscriber Identifier (IMSI), and thus limits the overhead per invocation to the length of an IMSI, i.e. 15 bits.

3 Enabling Mobile Web Services

There are plenty of business scenarios requiring to notify an end user's mobile device (e.g.[BIM+09]). Starting from notifications of events already scheduled to occur, like a deadline for an eBay bidding, reaching to scenarios that are not as predictable, e.g. long-term business transactions in the insurance economy as outlined in Section 4.

There exist two trivial solutions bypassing the aforementioned problem. The first trivial solution sends a message via SMS to initiate a callback from the cell phone to the enterprise system. SMS messages come with the drawback of being unreliable [MPD03, p. 743f]. Although, the SMS standard specifies a so-called *Message delivery report* and a *Message submission report*, the providers tend either not to implement these reports or they differ from the standard. Also, sending an SMS is expensive in terms of money/message compared to Internet traffic. Another trivial solution periodically polls on a Web Service provided by the enterprise system for new messages. However, the energy consumption and necessary network traffic rise with the polling interval.

Also, business processes based on the Business Process Modeling Notation (BPMN) involving mobile devices cannot be directly transformed into executable processes using a BPEL engine, if mobile devices cannot be notified of business events. When integrating heterogeneous systems, Web Services evolved as the state-of-the-art answer, and Web Service orchestration that integrates mobile devices [JMS05, AB06, SAD06] seems reasonable. Since Android does not offer support for providing Web Services, and existing third party Java libraries show compatibility issues, a Web Service Container for the Android platform is developed and used for developing the proposed solution illustrated in this paper.

Architecture In order to offer a solution for integrating mobile devices in terms of sending reliable, asynchronous notifications containing SOAP payload data, an architecture is introduced overcoming the problems when establishing inbound TCP connections to mobile devices. Fig. 1 shows the overall architecture. Besides a Web Service consumer that is part of an enterprise application, it consists of three components. The *Web Service Container* is, the first Web Service container for the Android platform. As the layer indicates,

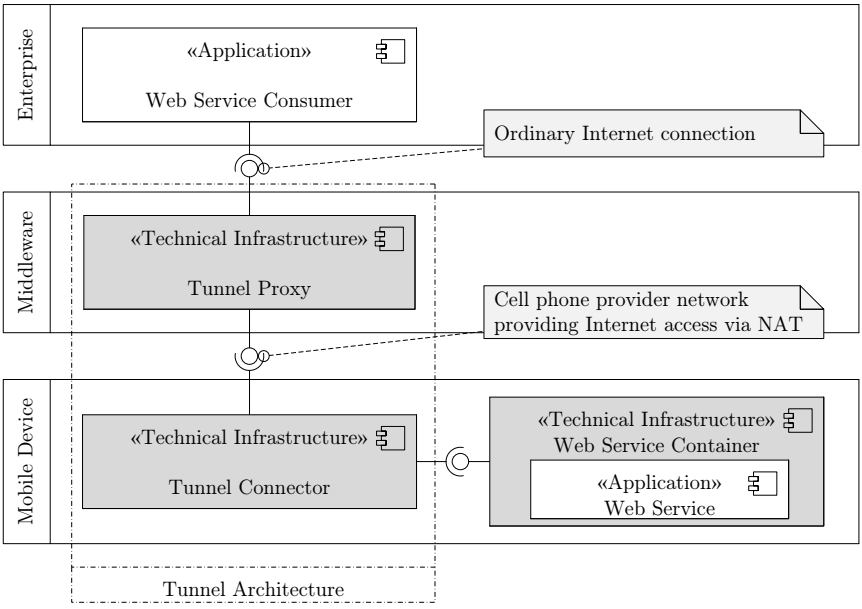


Figure 1: Communication platform architecture for push-based mobile Web Services

it runs on the mobile device. A *Tunnel Connector* connects to the Web Service Container component on demand. This component acts upon information indicated by the connection and is not required to analyse the actual payload. On initialisation, the Tunnel Connector establishes a new TCP/IP connection to the *Tunnel Proxy*. A Tunnel Proxy receives the actual SOAP request from a Web Service consumer. The mobile device is not directly accessible via TCP/IP, since only valid HTTP requests are forwarded to existing TCP/IP connections established by the Tunnel Connector. Loose coupling between the Tunnel Connector and the Web Service Container is worthwhile by the time future cell phone provider networks will use IPv6. Given that mobile devices get a (static) public address, the Web Services consumers can directly invoke the Web Service on the mobile device or cell phone. A migration then only requires to deactivate the Tunnel Connector and Tunnel Proxy components as well as they could coexist. This would have been impossible when integrating the Tunnel Connector functionality directly into the Web Service Container. Furthermore, the Tunnel Connector can be used on any platform, where existing Web Service containers or even application servers can be used to provide Web Services. The subsequent sections introduce the *Tunnel Architecture* components and their functionality in more detail.

Tunnelling mobile provider networks A mobile device connected to the Internet via a cell phone provider network uses mobile IPv4. Although there exist provider networks, which assign public IPv4 addresses [Ero08, p. 2], it is common practice for cell phone

provider network to dynamically assign private IP addresses. For accessing a mobile device behind the cell phone provider's NAT, the two components *Tunnel Connector* and *Tunnel Proxy* are used. The Tunnel Proxy is available for inbound connections on two TCP ports. While one port is used for Tunnel Connectors that establish persistent connections to the Tunnel Proxy, the other port is used for Web Service consumers to invoke Web Services on mobile devices. The architecture uses a one-to-many approach, such that the Tunnel Proxy has to be scalable, since it constitutes a potential bottleneck. Consequently, the Tunnel Proxy is designed as a scalable high performance component for handling numerous potential Web Service endpoints in idle mode while simultaneously being able to invoke mobile Web Service with a store and forward technique. In order to offer location transparency for mobile Web Services, each connection of a Tunnel Connector registers itself after connecting to the Tunnel Proxy. Obviously, the best possible identifier for a cell phone is the phone number. Unfortunately the phone number cannot be accessed on every Subscriber Identity Module (SIM). It strongly depends on the configuration of the cell phone provider network whether it is accessible or not. Anyhow, there are plenty of unique identifiers on a mobile device. While the International Mobile Equipment Identifier (IMEI) is assigned to each cell phone, the IMSI identifying the SIM is used to register new tunnel endpoints. Actually the phone number is resolved using the IMSI as an identifier within the cell phone provider network. An encryption and/or compression of the TCP/IP tunnel can be added to the architecture without affecting the way of enterprise integration or touching the implementation of the Web Service Container or Web Service consumer. In particular compression reduces traffic and increases throughput when connected via a cell phone provider network (see e.g. [Eri07]).

However, after registering to the Tunnel Proxy, a Tunnel Connector's connection can be used to invoke a mobile Web Service via a static endpoint. The Tunnel Proxy implements two HTTP methods. `GET` can be used combined with a `?wsdl` postfix to query the WSDL file of a Web Service on a mobile device. After querying the cell phone, the Web Service Container returns a WSDL file. Before forwarding this WSDL to the actual consumer, the SOAP address location has to be changed, such that the endpoint contains the hostname of the Tunnel Proxy instead of the mobile device. Further, the SOAP address location is enriched by the IMSI identifying each connected mobile device. This is the first part of the location transparency strategy that is reversed in case of an invocation. `POST` is used to invoke a mobile Web Service on a device. For the actual invocation of the Web Service, only the HTTP header information has to be analysed. Especially the `Content-Length` attribute is used to decide whether or not a SOAP response finished sending content. A Web Service request contains the IMSI number within the HTTP header, which extracted and used to identify the mobile device. Fig. 3 depicts the invocation of a mobile Web Service through the Tunnel Proxy. As illustrated, the original request is modified on a Web Service invocation such that the resulting HTTP header does not contain the IMSI number anymore, which causes the HTTP header to appear as an ordinary request to the Web Service Container. On incoming network traffic, the Tunnel Connector establishes a new connection to the Web Service Container and forwards the request. The Web Service response then is sent to the Tunnel Connector. If the HTTP `Content-Length` is reached, the Tunnel Proxy clears a tunnel connection. A *minimal encapsulation within TCP* prevents any payload overhead.

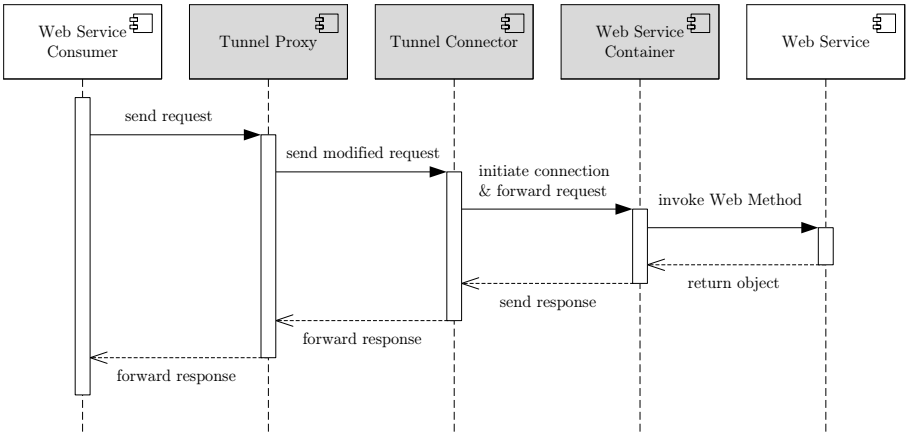


Figure 2: Push-based Mobile Web Service invocation

The Tunnel Proxy is designed to enable a high throughput using scalable concepts such as a pre-initialised thread pool. To deal with concurrent behaviour, Java's concurrency API is used to explicitly synchronise two threads. Actually, threads that process traffic and threads that administer new connections have concurrent access on a single selector in order to achieve a high degree of scalability. Thus, the Tunnel Proxy can use different thread priorities for accepting new connections and dispatching the processing of traffic, or even shut down one of both tasks to satisfy a service level agreement on peak load. By using a thread pool and combing it with the non-blocking I/O of Java NIO, the Tunnel Proxy reaches horizontal scalability even at reduced traffic (cf. 5).

Persistent TCP connections require to know when a device has been forcefully disconnected, e.g. when a system crashes. To detect half-open TCP connections, the TCP keep-alive option is used on the components Tunnel Connector and Tunnel Proxy. During development, different cell phone provider networks are tested. Concluding that cell phone providers tend to block TCP keep-alive packets, which impacts the solution. The unanswered TCP keep-alives forces the Tunnel Connector to close the connections, reconnect and re-register on the Tunnel Proxy, which results in an average network traffic of 350 bytes per two hours. However, using the TCP keep-alive technique does not require any awareness on the application level and is therefore a major design decisions. Further, TCP keep-alive scales [SV07], which is another reason to prefer using TCP keep-alive over sending heartbeat signals on the application level.

4 Application

As mentioned in Section 1, the related application scenario demonstrates an enterprise integration of the G1 cell phone. Therefore, a mobile application is integrated into a commercial claims management system of the insurance industry. The existing business process steps when reporting a claim are analysed and the business process is improved and enriched with IT artefacts. A holistic enterprise integration requires to asynchronously notify a customer of business events, e.g. for further inquiries of the insurance carrier or getting a customer's feedback after a claim is settled. By using push-based mobile Web Services, notifications of customers via mobile devices are possible. Initially, a mobile application enables insurance customers to report a minor automotive claim directly to the insurer. The first step of the related scenario is a customer reporting claim circumstances via a mobile application. Subsequently, the application offers location-based services, e.g. requesting a tow truck. The transmitted information, e.g. the location or pictures of the accident scene, is reviewed in the claims management enterprise system from the perspective of the claim personnel. Claims also contain information about automatically assigned business partners offering various third party services. In order to improve their customer service, it is of high relevance for insurance companies to collect feedback about customer satisfaction with the overall claims management process [Red98]. Therefore, the mobile claims assistance application allows insurance companies to request customer feedback from within their claims management systems. Whenever a claim is closed in the enterprise application, a business rule is triggered and a survey notification. is sent to the mobile phone where the user can launch the customer feedback screen by clicking on the expanded notification. The content of the feedback screen is dynamically sent to the mobile phone by invoking the mobile Web Service. The screen consists of a combination of text input fields and star bars to enable the rating of the overall satisfaction with the claims management process as well as to provide additional feedback. In addition, the customer can rate services provided by business partners such as repair shops or tow truck companies. The aggregation of this data in combination with the location information provides the insurance carrier with a holistic view of the service quality offered by third party business partners in different regions. The information can then be used to improve the overall service level for customers by cooperating with appropriate business partners. Furthermore, the customer feedback can be used to improve offered service, thereby leading to improved customer satisfaction and loyalty. A service is available for invocations, even if the user is currently using the device for other applications. Hence, an Android Background-Service is used to integrate the Web Service Container and the Tunnel Connector into a single always-on application.

5 Evaluation

The Tunnel Proxy uses the Java NIO API designed for high-performance, non-blocking, multiplexed I/O. Thus, for a benchmarking, the Tunnel Proxy is considered as a high-performance component. Consequently, a scalable approach is used for benchmarking

the component, which is introduced subsequently. Besides the Tunnel Proxy's scalability, for end users, the consumed energy of always-on applications is crucial, since it reduces the uptime of the mobile device and influences the battery lifetime. As a result, besides the Tunnel Proxy, the Web Service Container and the Tunnel Connector are evaluated concerning their additionally consumed energy.

Energy Consumption Haverinen et. al. show in [HSE07] that the consumed energy of always-on applications is a serious issue and keep-alive traffic increases the energy consumption of mobile devices.

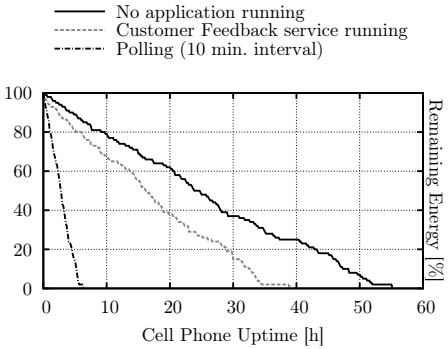


Figure 3: Energy consumption of the light-weight Web Service Container with running Tunnel Connector

Fig. 3 depicts the mobile device battery and the impact of the proposed solution. As one can see, the cell phone's uptime is reduced by approx. 30%, when running the prototypical application. However, the solution clearly outperforms a polling approach with an interval of 10 minutes. During the energy measurements the network traffic is sniffed. An analysis of the traffic shows that the cell phone is disconnecting every two hours due to unacknowledged TCP keep-alives. Concluding, that the results presented in this paper do not reflect an optimal, but rather a realistic setup.

Scalability As it is beyond manageable effort to run $10,000$ Android emulators at once in lab conditions, a simulator is developed that either acts as a Tunnel Connector or Web Service consumer. A LAN is used to simulate mobile device connections, such that the actual duration of a Web Service invocation is not evaluated, since mobile devices are commonly connected via a 3G network. Instead, the intrinsic processing time of the Tunnel Proxy is measured by performing a white-box test [FGPM08]. The Tunnel Proxy is executed on a Intel quad-core $2,4$ GHz, 2 GB RAM, running a Linux operating system with kernel version 2.6.28. The simulator application is deployed on 10 systems (cf. Fig. 4). In order to enable simultaneous connections with concurrent Web Service invocations, each simulator receives an UDP multicast packet containing instructions sent by a coordinator. Depending on an unique system identifier, the simulators execute the assigned instruction.

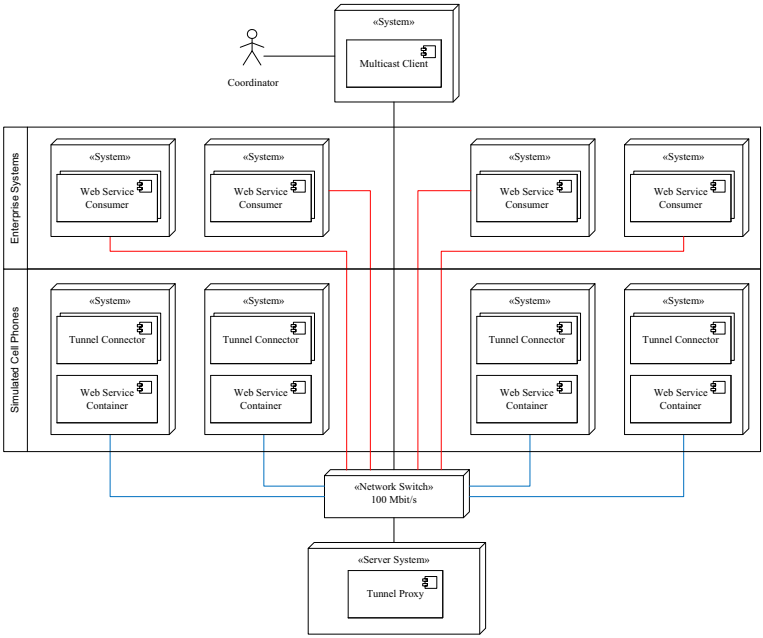
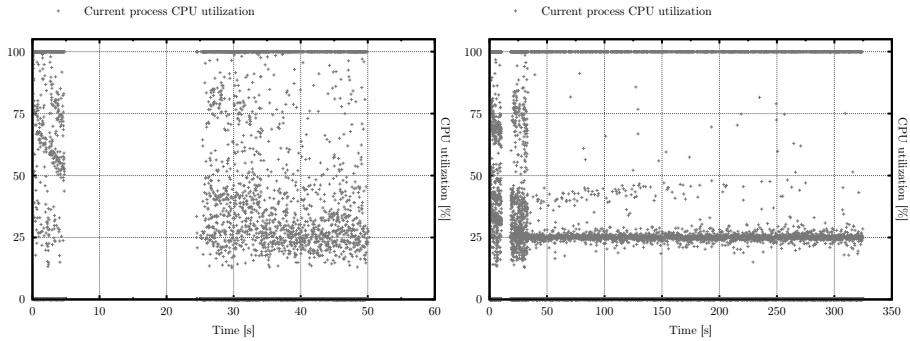


Figure 4: Setup of the testbed

For instance, *batch Web Service consumers* invoke n Web Services within a specified range of IMSI numbers with an interval i as a batch job. Another role simulates *concurrent Web Service consumers*, which concurrently invoke n Web Services on random mobile devices. In contrast to batch jobs, the intention of this scenario is to simulate an ad-hoc behaviour. To simulate random access, a specified range of IMSI numbers is used in an alternating order, such that concurrent threads invoke Web Services in an interval i with a random IMSI number. Also, the number of concurrent threads and invocations per thread can be specified. To initialise the test bed, some simulators have the role of *Web Service providers*, which run a single instance of the Web Service Container to save system resources, and n instances of the Tunnel Connector connecting to a specified Tunnel Proxy in an interval i. By combining these three roles, a variety of additional test cases become possible. For instance, $10,000$ concurrent invocations are simulated with access on random mobile devices, while another $1,000$ mobile devices simultaneously connecting to the Tunnel Proxy. Thus, the simulator can be applied with test scripts for larger, more complex scenarios.

Today's processor architectures require horizontal scalability that can be proved by visualising the distribution of the process' current CPU usage. On a quad-core machine, the maximum CPU usage is 400%. For a measurement the values have been normalised to a 100% scale. Thus, each time the process utilises the CPU below 25% uses at least one core. Values between 25% and 50% use at least two cores, between 50% and 75% at least three cores are used, and above 75% all four cores are used. Fig. 5(a) shows the results of $1,000$

concurrent Web Service invocations, while $1,000$ new simulated Tunnel Connectors are simultaneously connecting. The current CPU usage is a per process snapshot of the active processing time for the Tunnel Proxy and the elapsed time in nanoseconds (ns). Thus, the actual CPU utilisation is calculated based on the delta to the previous measurement point.



(a) Concurrent Random Access (b) Batch Job

Figure 5: Tunnel Proxy process CPU usage

The test is performed in multiple steps. First, simulated devices are connecting ($0 < s < 10$). Afterwards, the Web Service consumers invoke Web Services on $1,000$ simulated devices ($s > 20$), while $1,000$ new simulated Tunnel Connectors simultaneously connect. The normal distribution of the Tunnel Proxy's current CPU usage indicates a distribution on multiple CPU's that can be traced to a particular implementation detail. First, a thread pool is used guaranteeing an optimal CPU utilisation without producing scheduling overhead. Second, traffic is processed in so-called *handlers*, which are dispatched on a thread pool. Thus, any incoming request can become n handlers, executed on m cores or CPUs, which is enabled due to the combination of non-blocking `SocketChannels` of Java's NIO API with a thread pool. Alternative approaches would be either produce scheduling overhead (thread per request) on peak loads or just use a single core (single threaded with queuing). Fig. 5(b) shows the same test scenario as a batch job. One can clearly see the new connecting Tunnel Connectors ($20 < s < 30$), which are spread along the CPUs. Furthermore, the current process' CPU usage is either at 0%, 100% or 25%. While an utilisation of 0% is common, the 100% values can be traced back to the measurement methodology. Concluding, the values of the current process' CPU usage is strongly addicted to 25%, since each Web Service is invoked sequentially. Within the Tunnel Proxy, each new connection of a Tunnel Connector gets a buffer assigned, which is used to store, reassemble, and forward traffic. Two different buffer sizes, 8 KBytes and 512 bytes, are used within the Tunnel Proxy during the tests.

Fig. 6 illustrates the consumed memory for $10,000$ simultaneous connected Tunnel Connectors showing that reducing the buffer by a factor of 16 reduces the overall memory consumption only by a factor of 2. Given that the buffer size directly influences the Tunnel Proxy, e.g. number of reads to perform for each request, a larger buffer is to prefer. Further-

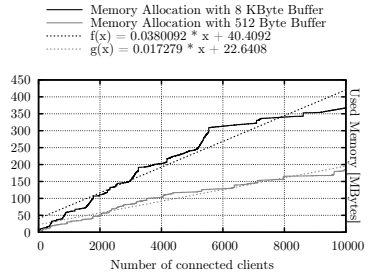


Figure 6: Memory consumption with different buffer sizes

more, the Tunnel Proxy uses hash-based search for an endpoint lookup, which is reflected by the memory-allocation characteristics, i.e. the re-hashes of the hashtables are visible. All measures confirm that the number of connected Tunnel Connectors does not influence the performance of the Tunnel Proxy during Web Service invocations.

6 Conclusion

This paper introduced an enabling technology in order gain access to provided mobile Web Services in today's cell phone provider networks. Based on the introduced access concept for mobile Web Service, a static endpoint can be used to query Web Services on mobile devices with dynamic IP addresses. Thus, they can be directly integrated into enterprise information systems, e.g. executable business processes. Feasibility has been shown by a prototypical implementation for the Android platform illustrated in Section 4. Section 5 analysed the additionally consumed energy, which is obviously relevant for end users, but rather figured out the potential for improvements. The solution consumes far less energy than a polling approach, which significantly distinguishes the proposed solution from other solutions. Further it has been shown that the Tunnel Proxy scales horizontally along with multiple cores when dealing with concurrent invocations and new connecting devices, and the responsible implementation is explained in detail. Although each component is scalable, future work will include a load balancing mechanism through system boundaries based on the HTTP protocol to serve numerous peers and increase fault tolerance, e.g. for system failures. When using push-based mobile Web Services, enterprises do not have to pay for expensive and unreliable SMS text messages to trigger mobile devices. In addition, business processes using the Business Process Execution Language can directly integrate mobile Web Services using the advantage of the Tunnel Architecture, e.g. location transparency. Since the middleware already has proven scalable behaviour, further research could benchmark the performance of the solution in terms of end-to-end communication of systems and mobile devices. Thus, the solution could be compared with alternative approaches, e.g. [LRF+09].

References

[AB06] Mustafa Adaçal and Ayşe B. Benner. Mobile Web Services: A New Agent-Based Framework. *IEEE Internet Computing*, 10(3):58–65, 2006.

[App10] Apple. *Apple Push Notification Service Programming Guide: Apple Push Notification Service*, 2010.

[BIM+09] Oliver Baecker, Tobias Ippisch, Florian Michahelles, Sascha Roth, and Elgar Fleisch. Mobile Claims Assistance. In *Proceedings of the 8th International Conference on Mobile and Ubiquitous Multimedia (MUM)*, 2009.

[Cod09] Google Code. *I-Jetty: webserver for the android mobile platform*, 2009.

[Eri07] Morgan Ericsson. The Effects of XML Compression on SOAP Performance. *World Wide Web*, 10(3):279–307, 2007.

[Ero08] Pasi Eronen. TCP Wake-Up: Reducing Keep-Alive Traffic in Mobile IPv4 and IPsec NAT Traversal. Technical Report 2, Nokia Research Center, January 2008.

[FC05] Patrick Farley and Matt Capp. Mobile Web Services. *BT Technology Journal*, 23(3):202–213, 2005.

[FGPM08] Chris Ford, Ido Gileadi, Sanjiv Purba, and Mike Moerman. *Patterns for Performance and Operability*. Auerbach Publications, Boston, MA, USA, 2008.

[HSE07] Henry Haverinen, Jonne Siren, and Pasi Eronen. Energy Consumption of Always-On Applications in WCDMA Networks. In *VTC Spring*, pages 964–968. IEEE, 2007.

[JMS05] François Jammes, Antoine Mensch, and Harm Smit. Service-oriented device communications using the devices profile for web services. In *MPAC '05: Proceedings of the 3rd international workshop on Middleware for pervasive and ad-hoc computing*, pages 1–8, New York, NY, USA, 2005. ACM.

[KL07] Yu-Kwong Ricky Kwok and Vincent K.N. Lau. *Wireless Internet and Mobile Computing: Interoperability and Performance (Information and Communication Technology Series,)*. Wiley-Interscience, 2007.

[LRF+09] Francisco Milagro Lardies, Pablo Antolin Rafael, Joao Fernandes, Weishan Zhang, Klaus Marius Hansen, and Peeter Kool. Deploying Pervasive Web Services over a P2P Overlay. In *WETICE '09: Proceedings of the 2009 18th IEEE International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises*, pages 240–245, Washington, DC, USA, 2009. IEEE Computer Society.

[MPD03] S. A. M. Makki, Niki Pissinou, and Philippe Daroux. Mobile and wireless Internet access. *Computer Communications*, 26(7):734–746, 2003.

[PvdH07] Mike P. Papazoglou and Willem-Jan van den Heuvel. Service Oriented Architectures: Approaches, Technologies and Research Issues. *VLDB Journal*, 16(3):389–415, 2007.

[Red98] Thomas C. Redman. The impact of poor data quality on the typical enterprise. *Commun. ACM*, 41(2):79–82, 1998.

[Rot09] Sascha Roth. A Scalable Communication Platform for Push-based Mobile Web Services. Master's thesis, Hochschule Darmstadt – University of Applied Sciences (Department of Computer Science), 2009.

[SAD06] Daniel Schall, Marco Aiello, and Schahram Dustdar. Web services on embedded devices. *IJWIS*, 2(1):45–50, 2006.

[SV07] Konstantin Shemyak and Kai Vehmanen. Scalability of TCP Servers, Handling Persistent Connections. In *ICN '07: Proceedings of the Sixth International Conference on Networking*, page 89, Washington, DC, USA, 2007. IEEE Computer Society.

[Wir01] Wireless Application Protocol Forum, Ltd. *Push Access Protocol*, 2001.