[RBHSt06] *Nicolas Repp, Rainer Berbner, Oliver Heckmann, Ralf Steinmetz;* **A Cross-Layer Approach to Performance Monitoring of Web Services;** Proceedings of the Workshop on Emerging Web Services Technology (in conjunction with IEEE ECOWS 2006), CEUR-WS, Zürich, Dezember 2006, S. 17-27.

A Cross-Layer Approach to Performance Monitoring of Web Services

Nicolas Repp, Rainer Berbner, Oliver Heckmann, and Ralf Steinmetz

Technische Universität Darmstadt
Multimedia Communications Lab (KOM)
Merckstrasse 25, 64283 Darmstadt, Germany
repp@kom.tu-darmstadt.de

Abstract. An increasing amount of applications are currently built as Web Service compositions based on the TCP/IP+HTTP protocol stack. In case of any deviations from desired runtime-behavior, problematic Web Services have to be substituted and their execution plans have to be updated accordingly. One challenge is to detect deviations as early as possible allowing timely adaption of execution plans. We advocate a cross-layer approach to detect bad performance and service interruptions much earlier than by waiting for their propagation through the full protocol stack.
This position paper describes an approach to gain detailed real-time information about Web Service behavior and performance based on a cross-layer analysis of the TCP/IP+HTTP protocols. In this paper we focus especially on TCP. The results are used to make decisions supporting service selection and replanning in service-oriented computing scenarios. Furthermore, generic architectural components are proposed implementing the functionality needed which can be used in different web-based scenarios.

1 Introduction

Almost every Internet user has encountered problems while using services in the Internet, e.g., browsing the World-Wide Web or using Email. Long to infinite response times due to congestion or connection outage, non-resolvable URLs, or simple file-not-found errors are some of the most common ones. Human users tend to be flexible in case of any service "misbehavior". Users wait and check back later or even select a different service if the originally requested service is not available. In contrast, computer systems as service consumers are not as flexible. Appropriate strategies to handle those runtime events have to be implemented during design time of the computer system.

Services are the key building block of service-oriented computing. A service is a self-describing encapsulation of business functionality (with varying granularity) according to [1]. Following the service-oriented computing paradigm, applications can be assembled out of several independent, distributed and loosely-coupled services [2]. Those services can be provided even by third parties. One option to implement services from a technical perspective is the use of Web Services. Web Services are based on different XML-based languages for data exchange and interface description, e.g., SOAP and the Web Service Description Language (WSDL). For the transport of data and the

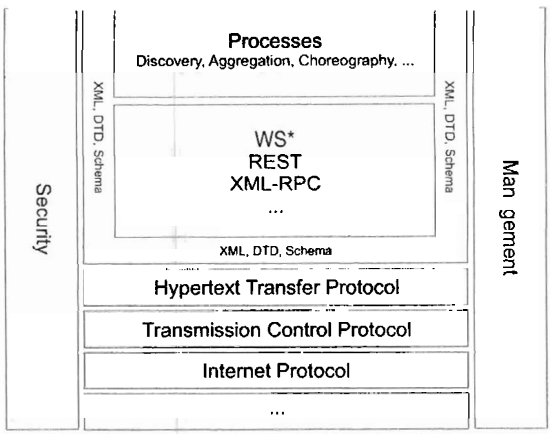

Fig. 1. Modified W3C Web Services Architecture Stack [3]

Web Service invocation mainly the Transmission Control Protocol (TCP) / Internet Protocol (IP) suite (e.g., RFC 793, [4], or [5]) as well as the Hypertext Transfer Protocol (HTTP - e.g., RFC 2616 or [6]) are used. Figure 1 shows the W3C Web Services Architecture Stack enhanced by alternative Web Service technologies and the communication protocols used. It will be the basis for our further considerations.

In order to build applications from different existing Web Services the following generic phases are needed [7]: First, suitable Web Services have to be selected according to the functional and non-functional requirements of the application. Second, the selected Web Services have to be composed to an execution plan. Hereto, a composition can be described, e.g., on basis of the Business Process Execution Language (BPEL) [8]. In the next step the execution plan can be processed. During the execution phase it is possible that parts of the composition do not act as expected with regard to the non-functional requirements. Reasons for misbehavior of Web Services are manyfold, e.g., server errors while processing a request, network congestion or network outages. Therefore, it is necessary to select alternative Web Services and to replan the Web Service execution [9]. Replanning is always a trade-off between the costs of creating new plans to fulfill the overall non-functional requirements and the costs of breaking the requirements [10]. Timely action is required to reduce the delay in the execution of an application due to replanning and substitution of Web Services. Hence, we propose a proactive approach initiating countermeasures as soon as there is evidence that a deviation might occur in the near future with a certain probability p. To start replanning before the deviation happens allows replanning to be carried out in parallel to the service execution itself. The results of replanning have to be discarded with probability $1 - p$ as the alternative plans are not needed.

Furthermore, current approaches often lack detailed information about the status of a Web Service due to the information hiding implemented in the layer model of the TCP/IP+HTTP protocol stack underlying Web Services. For this, we advocate a cross-layer approach to detect bad performance and service interruptions. Cross-layer analysis allows decisions based on deeper knowledge of the current situation as well as decisions made much earlier than by waiting for information propagating through the full protocol stack.

The rest of this position paper is structured as follows. In the next section we describe Quality-of-Service (QoS) and its meaning for Web Services. We especially focus on performance as a part of Web Service QoS. Afterwards, the relation between TCP/IP+HTTP and Web Service performance is discussed. Our cross-layer approach to performance monitoring an performance anomaly detection of Web Services is introduced thereafter. The paper closes with a conclusion and an outlook on future work.

2 Quality-of-Service and Performance of Web Services

In this section we discuss QoS with regard to Web Services and Web Service compositions with a focus on Web Service performance.

2.1 Quality-of-Service with regard to Web Services

Similar to QoS requirements in traditional networks, there is a need to describe and manage QoS of Web Services and Web Service compositions. Generally, QoS defines non-functional requirements on services independent from the layer they are related to. QoS can be divided into measurable and non-measurable parameters. The most common measurable parameters are performance-related, e.g., throughput, response time, and latency. Additionally, parameters like availability, error-rate, as well as various non-measurable parameters like reputation and security are of importance for Web Services [10] [11]. The meaning of QoS requirements can differ between service providers and service requesters in a service-oriented computing environment [11]. From a service providers' perspective, providing enough capacity with the quality needed to fulfill Service Level Agreements (SLA) with different customers is a core issue. Service requesters are more focused on managing bundles of Web Services from different vendors in order to implement their business needs. Therefore, management of QoS requirements is done on aggregations of Web Services, to a lesser extend on single Web Services.

There is a variety of other definitions of Web Service QoS. A more extensive approach identifies the following requirements [12]: performance, reliability, scalability, capacity, robustness, exception handling, accuracy, integrity, accessibility, availability, interoperability, security, and network-related QoS requirements. Especially the last requirement is of further interest. As many requirements of Web Service QoS are directly related to the underlying network and its QoS, implementations of network QoS mechanisms, e.g., Differentiated Services (DiffServ) or the Resource Reservation Protocol (RSVP), are also covered by the definition as well.

2.2 Performance of Web Services

Performance of Web Services is not a singular concept. Rather, it consists of several concepts which themselves are connected to different metrics and parameters. Again, there are several definitions of Web Service performance. We will use the definition provided by the Web Services Architecture Working Group of the W3C as a foundation for our own defintion. According to the W3C, performance is defined in terms of throughput, reponse time, latency, execution time, and transaction time [12]. Both execution time and latency are sub-concepts of the W3Cs definition of response time. Transaction time describes the time needed to process a complete transaction, i.e., an interaction consisting of several requests and responses belonging together.

For this paper, we define performance in terms of throughput and response time. Response time is the time needed to process a query, from sending the request until receiving the response [13]. Response time can be further divided into task processing time, network processing time, i.e., time consumed while traversing the protocol stacks of source, destination, and intermediate systems, as well as network transport time itself. In case of an error during the processing of a request or a response, the response time measures the time from a request to the notification of an error. We define response time as follows:

$$t_{response}(ws) = t_{task}(ws) + t_{stack}(ws) + t_{transport}(ws)$$

A large fraction of a web service's response time is determined by the processing time for requests and their respective messages in both intermediate systems and end-points. For the measurement of the response time, the encapsulation of data into XML messages and vice versa, compression and decompression of data, as well as encryption and decryption of messages also have to be taken into account. Furthermore, time for connection setup, for the negotiation of the connections parameters as well as the amount of time used for authentication are part of the response time as well.

Throughput, measured in connections, requests or packets per second, describes the capability of a Web Service provider to process concurrent Web Service requests. Depending on the layer, different types of connections can be the basis for measurements, e.g. TCP connections, HTTP connections, or even SOAP interactions. We define the throughput of a Web Services as:

$$throughput(ws) = \frac{\#requests(ws)}{time}$$

Additionally, we have to define the concept of "performance anomaly" we will use later on. Performance anomalies describe deviations from the performance expected in a given situation. Performance anomalies do not have to be exceptions or even errors, e.g., a response time which exceeds the value defined in a SLA is also a performance anomaly with regard to business requirements. Furthermore, performance better than expectations is also an anomaly.

3 A Cross-layer Approach to Performance Monitoring and Anomaly Detection

In this section we describe an approach for performance monitoring and performance anomaly detection based on packet capturing and the application of simple heuristics. Therefore, we analyze IP, TCP, and HTTP data. The analysis of SOAP is not in scope of this paper, as we want to stay independent of a certain Web Service technology. Our approach can be applied to various alternative Web Service technologies as well, e.g., XML-Remote Procedure Call (XML-RPC) or Representational State Transfer (REST). Nevertheless, in our examples we use SOAP as it is the most common Web Service technology in use.

3.1 Protocol Parameters for Performance Monitoring

Consider the simple Web Service invocation of a single Web Service as depicted in Figure 2. A service requester generates a SOAP request and sends the message using HTTP to the service provider for further processing. The message has to pass several intermediate systems on its way between the interaction's endpoints. The SOAP response message is again transported using HTTP.

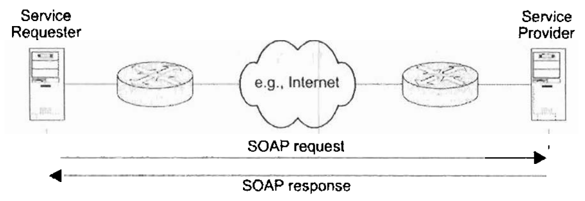

Fig. 2. Simple Web Service interaction

During data transfer several problems can occur, which all have an impact on Web Service execution. Beginning with the network layer, we may face routing problems, e.g., hosts which are not reachable, congestion in Internet routers as well as traffic bursts. Additionally, on transport layer there are also potential pitfalls like the retransmission of packets due to packet loss or connection setup problems generating delays. Finally, there are also some potential problems on application layer with regard to Web Services for example in form of resources, which are not existing or not accessable for HTTP or problems in processing of SOAP messages due to incomplete or non-valid XML data.

Although, many of the above problems are solved in modern protocol stack implementations, we can use the knowledge about them to define measurement points for performance monitoring. Depending on the problems in scope different protocol parameters have to be used. Table 1 gives an overview of measurement points on different protocol layers. We will use the transport layer parameters as an example to derive metrics and heuristics for performance anomaly detection in the following section.

Protocol	Measuring Point / Parameter
IP	ICMP messages
TCP	Size of advertising window
	Roundtrip time (RTT)
	Sequence numbers in use
	Flags used in packets
	Information about timers
HTTP	Header information

Table 1. Measuring points per protocol layer

3.2 Metrics and Heuristics for Performance Anomaly Detection

As noted in Section 2.1 we can differentiate between the requirements of service requesters and service providers. To visualize our concepts we will focus on the service requester's perspective in this position paper. Before basic heuristics are proposed we present metrics based on the parameters presented in Table 1, which will be the foundation of our heuristics. We propose several metrics based on parameters of the transport layer protocol:

- *M1* - Average throughput in bytes per second (BPS).
- *M2* - Throughput based on a moving average over window with size *n* seconds in BPS.
- *M3* - Throughput based on exponential smoothing (first degree) with α varying in BPS.
- *M4* - Roundtrip time based on a moving average over window with size *n* segments in seconds per segment.
- *M5* - Number of gaps in sequence numbers based on a moving average over window with size *n* seconds in number of gaps per second.

The aggregation of single metrics in combination with the usage of appropriate thresholds allows us to build heuristics in order to detect anomalies with performance impact. The following two simple heuristics show the idea how to design heuristics based on the metrics discussed. Both were derived from experimentations in our Web Service test environment.

- $H1_{Requester}$: *M1* (or *M2*, *M3*) in aggregation with *M4*, i.e., throughput combined with RTT.
- $H2_{Requester}$: *M4* in aggregation with *M5*, i.e., RTT combined with the amount of gaps in TCP sequence numbers.

Singular metrics are in some cases not sufficient for robust monitoring, e.g., *M5* without any information about RTT does not offer useful information.

In addition to those transport layer based heuristics, further parameters from other protocol layers and the respective metrics can be combined in order to create different cross-layer heuristics. Nevertheless, it is important that metrics and the related heuristics have to be calculated in an efficient way in order to keep additional processing times of our approach low.

3.3 Exemplary Evaluation of Our Approach

To show the feasibility of our approach we set up an experiment. The test environment consists of a 1.4 GHz Centrino with 1.256 GByte RAM running Windows XP as service requester and a 1.42 GHz G4 with 1 GByte RAM running Mac OS X as service provider. Apache Tomcat 5.5.17 is used as an application server. Both systems use Java 1.5 and Axis 1.4 as SOAP implementation. They are connected by 100 MBit/s ethernet. For packet capturing windump v3.9.3 is used.

First, we measure the response time of a Web Service in our test environment. As payload we use SOAP messages of variable size. Table 2 shows the results of measuring 20 individual runs both with and without network outage for a payload of 20 MByte, a test scenario, which was already implemented in our test environment. Similar results can be observed with a payload of 150 KB. Network outages are equally distributed

$t_{response}(ws)$ [ms]	minimum	maximum	average
w/o outage	8,743	9,604	8,891
w/ outage	601,204	605,831	604,186

Table 2. SOAP response times

in the interval [0;max($t_{response}(ws)$ w/o outage)]. A network outage is modelled as a permanent 100% packet loss, i.e., without a restart of the connection. Other scenarios, e.g., varying or temporary packet loss, are not in focus of this position paper. As Table 2 shows, the response time of our Web Service varies between 8.9 seconds (without outage) and 10.07 minutes (with outage) for a 20 MByte payload.

rtt [ms]	minimum	maximum	average
$H1_{Requester}$	0.22	0.41	0.31

Table 3. Roundtrip times

In a next step, we apply $H1_{Requester}$ on our sample with network outages. Especially the roundtrip time extracted from TCP packets can be used as trend estimate for the overall response time in our scenario. Table 3 shows the average roundtrip times of all 20 runs. Using a moving average of the roundtrip times measured as a benchmark for the roundtrip time of the packet in transfer, a warning to the replanning system can be sent, e.g., if the estimated time (or a multiple) is exceeded twice or more in a row. Unfortunately, throughput was not as good as the RTT as an indicator for performance anomalies in the given scenario.

3.4 Identification of Required Architectural Components

In order to implement our ideas several architectural components are needed. The key building blocks are depicted in Figure 3.

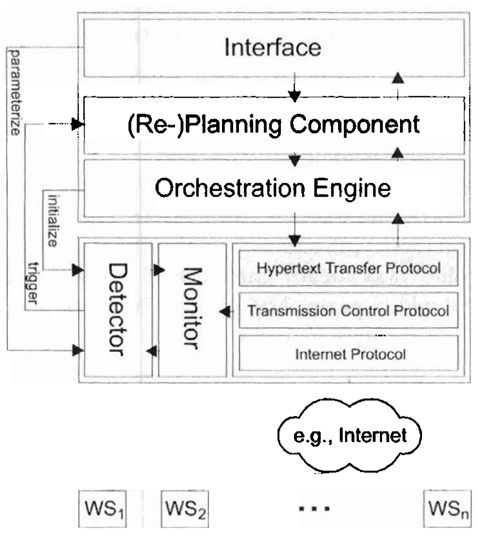

Fig. 3. Proposed architectural components

The upper part of Figure 3 describes existing generic components used for planning and executing of Web Service compositions. The *Interface* allows deployment of workflows and configuration, the *(Re-)Planning Component* generates and adapts execution plans, which are thereafter executed by an *Orchestration Engine*. We propose the use of our Web Service Quality-of-Service Architectural Extension (WSQoSX) as implementation means for the functionality needed. WSQoSX already supports planning and replanning of compositions [7] [10].

The lower part of the figure describes the two core components of our approach in addition to the protocol stack. This enhanced architectural blueprint is named Web Service - Service Monitoring Extension (WS-SMX). The *Monitor* specifies a component capable of eavesdropping of the network traffic between service requester and provider. It also implements pre-filtering of the data passing by reducing it to the protocol data of interest. Its data is passed to a *Detector* component, which is responsible for the data analysis and therefore the performance anomaly detection. The *Detector* component will implement the heuristics discussed in Section 3.2. The *Orchestration Engine* initializes the *Detector*, which itself prepares the *Monitor*. The *Detector* analyses the data received by the *Monitor* and triggers the *(Re-)Planning Component* in case of any critical findings. Additionally, the *Detector* component can be configured using the *Interface*. Both *Monitor* and *Detector* are implemented in a first version in our test environment based on Java 1.5 in combination with libpcap for packet capturing.

4 Related Work

As our approach is based on research of various domains this section gives an overview of related work in those domains. Gschwind et al. [14] describe WebMon, a performance analysis system with focus on Web transactions, i.e. transactions between a Web browser and a Web server. Monitoring is done on basis of HTTP. Web Services as remote method invocations as well as a further processing of the results of the analysis are not in scope of their paper. Similar mechanisms as the ones proposed by us are implemented in the commercial software package VitalSuite by Lucent, which is used for capacity planning and QoS management in large networks. VitalSuite can also analyze different protocol layers simultaneously. In contrast to the system we propose, VitalSuite's focus is on reporting for end-users instead of automated management. A more detailed view on performance management of Web Services is discussed by Schmietendorf et al. [15]. The Web Services Trust Center (WSTC) allows Web Services to be registered at and measured by an independent third party for SLA management. WSTC enables the monitoring of performance and availability of Web Services, but not under real-time requirements.

The management of Web Service compositions, their orchestration as well as their optimization and planning is emphasized in various papers, partly mentioned in the introduction. Of further interest in that domain is the Web Service Manager (WSM) introduced by Casati et al. [16] focusing on the business perspective of Web Service management, e.g., detecting and measuring SLA violations.

Fundamental work in the area of packet capturing, its justification and optimization was carried out e.g., by Feldmann [17] and Mao et al. [18]. Both do not focus on potential areas of application for packet capturing but on measurement itself. Feldmann uses cross-layer capturing and analysis of TCP and HTTP for later Web performance studies. Mao et al. describe both drawbacks and advantages of performance analysis of Web applications based on packet capturing mechanisms. Furthermore, a reliable and efficient approach for monitoring in distributed systems based on dispatching is discussed.

The idea of anomaly detection to predict certain critical situations is already used, e.g., in the area of network security, especially in network intrusion detection. Mainikopoulos et al. describe the use of statistical methods applied to network usage traces for anomaly detection, e.g., an attack on a networked system [19]. Another area of application is discussed by Yuan et al. [20]. They propose a system for automated problem diagnosis in applications based on system event traces. The correlation of current traces and patterns of well known problems allows an automatic identification of problem sources and prediction of possible system errors. Furthermore, the authors use statistical learning and classifying methods to dynamically adapt and improve their system.

5 Conclusion and Future Work

In this position paper we show that it can be beneficial to use information gathered on different protocol layers for decision support. We present an approach and several architectural components, which use hidden, low layer technical information for proactive replanning of Web Service compositions. As this is a position paper there are still some open issues we are researching. We are currently testing machine learning algorithms for anomaly detection. Furthermore, we are working on enhancements of existing optimization models for Web Service compositions to support replanning [10]. Additionally, we will test our approach from a service requester's perspective in real world scenarios, using Web Services available to the public, e.g., from Amazon or via Xmethods.

Using our approach for proactive replanning is not limited to SOAP Web Services. As we are collecting our data on lower layers, the type of Web Service can be exchanged, e.g., REST and XML-RPC based Web Services can also be supported. But we are not even limited to Web Services as an area of application. The approach can be of benefit, e.g., to enhance Web browsers to detect network problems in a faster way.

Acknowledgments

This work is supported in part by E-Finance Lab e.V., Frankfurt am Main.

References

1. Papazoglou, M.P.: Service-oriented computing: Concepts, characteristics and directions. In: Proceedings of the Fourth International Conference on Web Information Systems Engineering (WISE03). (December 2003) 3–12
2. Bichler, M., Lin, K.J.: Service-oriented computing. IEEE Computer **39**(3) (March 2006) 99–101
3. Booth, D., Haas, H., McCabe, F., Newcomer, E., Champion, M., Ferris, C., Orchard, D.: Web services architecture (2004. http://www.w3.org/TR/ws-arch/, accessed: 2006/07/02)
4. Stevens, W.R.: TCP/IP illustrated (vol. 1): the protocols. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (1994)
5. Tanenbaum, A.S.: Computer Networks, Fourth Edition. Prentice Hall, Indianapolis, Indiana, USA (August 2002)
6. Mogul, J.C.: Clarifying the fundamentals of http. In: WWW '02: Proceedings of the 11th international conference on World Wide Web. (May 2002) 25–36
7. Berbner, R., Grollius, T., Repp, N., Heckmann, O., Ortner, E., Steinmetz, R.: An approach for the management of service-oriented architecture (soa) based application systems. In: Proceedings of the Workshop Enterprise Modelling and Information Systems Architectures (EMISA 2005). (October 2005) 208–221
8. Curbera, F., Khalaf, R., Mukhi, N., Tai, S., Weerawarana, S.: The next step in web services. Commun. ACM **46**(10) (2003) 29–34

9. Canfora, G., Penta, M.D., Esposito, R., Villani, M.L.: Qos-aware replanning of composite web services. In: Proceedings of the IEEE International Conference on Web Services (ICWS'05). (July 2005) 121–129
10. Berbner, R., Spahn, M., Repp, N., Heckmann, O., Steinmetz, R.: An approach for replanning of web service workflows. In: Proceedings of the 12th Americas Conference on Information Systems (AMCIS'06). (August 2006)
11. Menascé, D.A.: Qos issues in web services. IEEE Internet Computing **6**(6) (2002) 72–75
12. Lee, K.C., Jeon, J.H., Lee, W.S., Jeong, S.H., Park, S.W.: Qos for web services: Requirements and possible approaches (2003. http://www.w3c.or.kr/kr-office/TR/2003/ws-qos/, accessed: 2006/07/03)
13. Jain, R.: The art of computer systems performance analysis: techniques for experimental design, measurement, simulation, and modeling. John Wiley & Sons, Inc., New York, NY, USA (1991)
14. Gschwind, T., Eshghi, K., Garg, P.K., Wurster, K.: Webmon: A performance profiler for web transactions. In: Proc. of the 4th IEEE Int'l Workshop on Advanced Issues of E-Commerce and Web-Based Information Systems - WECWIS 2002. (June 2002) 171–176
15. Schmietendorf, A., Dumke, R., Stojanov, S.: Performance aspects in web service-based integration solutions. In: Proc. of the 21st UK Performance Engineering Workshop - UKPEW2005. (July 2005) 137–152
16. Casati, F., Shan, E., Dayal, U., Shan, M.C.: Business-oriented management of web services. Commun. ACM **46**(10) (2003) 55–60
17. Feldmann, A.: Blt: Bi-layer tracing of http and tcp/ip. Comput. Networks **33**(1-6) (2000) 321–335
18. Mao, Y., Chen, K., Wang, D., Zheng, W.: Cluster-based online monitoring system of web traffic. In: WIDM '01: Proceedings of the 3rd international workshop on Web information and data management. (November 2001) 47–53
19. Manikopoulos, C., Papavassiliou, S.: Network intrusion and fault detection: a statistical anomaly approach. IEEE Communications Magazine **40**(10) (October 2002) 76–82
20. Yuan, C., Lao, N., Wen, J.R., Li, J., Zhang, Z., Wang, Y.M., Ma, W.Y.: Automated known problem diagnosis with event traces. In: Proceedings of EuroSys2006. (April 2006) 375–388