Enabling Publish / Subscribe with Cots Web Services across Heterogeneous Networks

Espen Skjervold, Trude Hafsøe, Frank T. Johnsen and Ketil Lund

Norwegian Defence Research Establishment, Instituttveien 20, 2007 Kjeller, Norway

Abstract. In scenarios such as search-and-rescue operations, it may be required to transmit information across multiple, heterogeneous networks, often experiencing unreliable connections and limited bandwidths. Typically, there will be traffic within and across radio networks, as well as back to a central infrastructure (e.g., a police command post) when a reach-back link is available. This implies that using Publish/Subscribe is advantageous in order to reduce network traffic, and that store-and-forward capabilities are required to handle the instability of radio networks. At the same time, it is desirable to use commercial software based on standards as far as possible, in order to reduce cost and development time, and to ease interconnection of systems from different organizations. We therefore propose using SOA based on Web services in such scenarios. Indeed, Web services are targeted at stable, high-speed networks, but our work shows that such usage is feasible. In this paper, we add Publish/Subscribe functionality to standard, unmodified Web services through the use of our prototype middleware solution called the Delay and Disruption Tolerant SOAP Proxy (DSProxy). In addition to the ability to make Web services delay and disruption tolerant, the DSProxy enables SOAs in scenarios as described above. The DSProxy has been tested in field trials, with promising results.

1 Introduction

Commercial off-the-shelf (COTS) Web services are generally based on Request/Response (client-server) mechanisms [5]. However, many systems, environments, and situations could benefit from using the Publish/Subscribe paradigm instead, which is characterized by scalability, decoupled communication peers and asynchronous communication. In particular, Publish/Subscribe is essential to support mobile ad-hoc networks (MANETs), i.e., dynamic collections of nodes with rapidly changing multi-hop topologies that are composed of wireless links [7].

In search-and-rescue operations, it may be required to transmit information between many or all participants, and this can require traversing multiple heterogeneous networks. In order to enable different organizations running systems developed by different vendors to interoperate, it is crucial to base such communications on open and widely accepted standards. Considering the ubiquity of standard Web services and the usefulness of Publish/Subscribe, bringing the two together would provide important benefits, as organizations and enterprises can leverage the power of Publish/ Subscribe mechanisms with their existing Web services and clients.

Web services technology is mostly associated with the traditional client-server paradigm. As pointed out by (Vinoski, 2004), this scheme is generally much less efficient than push-based communication such as Publish/subscribe, and with OASIS' WS-Notification and W3C's WS-Eventing, the Publish/Subscribe paradigm has entered the Web service arena.

WS-Notification is a standard organized in a group of specifications that enable Publish/Subscribe-based communication between Web services. It comprises WS-BaseNotification, WS-BrokeredNotification and WS-Topics. While WS-BaseNotification defines which interfaces consumers (clients) and producers (servers) should expose, WS-BrokeredNotification introduces the concept of a message broker, an intermediary node which decouples consumers and publishers, and relieves producers from several tasks associated with Publish/Subscribe. WS-Eventing basically defines functionality similar to WS-BaseNotification, but with the addition of the Subscription Manager role, which enables subscription-related tasks to be handled by other nodes than the producer.

While WS-Notification and WS-Eventing offers standards-based Publish/Subscribe using Web services, they require the introduction of supporting frameworks, and new Web services and Web service clients that adhere to the standards must be developed. Standard Web service clients normally create outbound connections to Web services using HTTP and TCP, and since they typically do not run inside application servers, they have no way of listening for incoming connections. Similarly, ordinary Web services typically do not initiate outbound connections, and must be replaced by Publish/Subscribe-capable Web services. Furthermore, while we expect industry support for WS-Notification and WS-Eventing to mature in the future, it currently seems not to be a large selection of supported commercialized implementations available. Finally, it should also be noted that when services based on WS-Notification or WS-Eventing send notifications to subscribers, they do so by setting up an individual point-to-point connection to each subscriber. This means that these mechanisms have an untapped potential for efficiency improvement with respect to network traffic.

We have addressed these challenges using a middleware approach. Our primary goals have been 1) to enable Publish/Subscribe for existing standard Web services and clients without having to rewrite any software; and 2) to enable such Publish/Subscribe Web services to traverse multiple heterogeneous networks.

The result is a lightweight, prototype middleware system, called the Delay and Disruption Tolerant SOAP Proxy (DSProxy), which is able to meet the challenges described above. The DSProxys form an overlay network, which hides network heterogeneity and instability from the Web services and the clients. Between the DSProxys, optimizations such as data compression are used to reduce overhead, and several different transport-protocols are available, for handling different types of networks.

Externally, the DSProxy middleware solution is compatible with standard Web services by employing communication based on SOAP over HTTP/TCP. This means that existing Web services can be used together with the DSProxy overlay network entirely without modification, while clients only need to replace the URL that addresses the services.

This solution also means that the DSProxy overlay network can be deployed only where needed; it is not an all-or-nothing solution that must be deployed everywhere. Typically, a DSProxy overlay network will be deployed within a MANET and between the MANET and a fixed network, while it is not needed internally in the fixed network. Clients in the fixed network will still be able to invoke services in the MANET and vice versa.

The remainder of this paper is structured as follows: In Section 2 we present related work, before describing the design and principles of the DSProxy in Section 3. We then present the configuration and results from a large field trial where the DSProxy was tested in Section 4, before concluding in Section 5.

2 Related Work

Publish/Subscribe systems have been around for a long time, and one of the earliest publicly described Publish/Subscribe systems was reportedly the “news” system of the Isis Toolkit, described at the 1987 ACM Symposium on Operating Systems Principles conference [2]. There exist many systems that support Publish/Subscribe-based information dissemination, ranging from open-source projects to commercial Enterprise Service Buses (ESBs). Some of the most well known middleware systems are Apache ActiveMQ, OMG Corba Event Service and Notification Service, IBM WebSphere MQ, and Real-Time Innovations (RTI). In ActiveMQ, the Publish/Subscribe functionality is built on top of Java Messaging Queues (JMSs). RTI is based on DDS, which is an open middleware specification for enabling Publish/Subscribe communications in real-time and embedded systems. While JMS-based solutions are message-centric, DDS and RTI differ by being data-centric, and in addition to topics offer keys which uniquely identifies objects. While offering Publish/Subscribe, these solutions all require developers to create custom applications utilizing the provided APIs.

The work by [1] points out that a scalable and efficient approach for achieving event dissemination in Publish/Subscribe systems is to employ an overlay network of brokers. In their work, they attempt to reorganize the overlay to reduce overhead associated with event dissemination. They point out that an alternative way to achieve efficient event dissemination is to use smart dissemination algorithms that avoid flooding events on the overlay, but they do not implement this. Also, their solution, named SIENA, is based on a proprietary, experimental Publish/Subscribe system. In our work, we do not want frequent overlay reconfiguration if we can avoid it, due to the overhead associated with management traffic. Thus, we use the complementary approach of smart dissemination algorithms. Further, we base our implementation on open Web services standards rather than a proprietary API, making our solution usable for a broader range of client applications and services. Another distinction from the work of [1] is that while SIENA is a content based Publish/Subscribe system, Web service Publish/Subscribe specifications adhere to a topic based scheme.

[4] have created a distributed event notification system (DENS) for MANETs. They developed a fuzzy logic based subscription language allowing expressive subscriptions and sophisticated event filtering. DENS is delay tolerant, an important feature in dynamic environments such as MANETs, and it builds an overlay, which performs store-and-forward of event messages. This solution is implemented and evaluated in a network emulator. The solution is shown to function well, proving that building a store-and-forward overlay for event notification in a MANET is both feasible and efficient. In our work we leverage this knowledge, by making the DSProxy system create an overlay network for event notification in MANETs. However, we also introduce the capability of configuring static routes for some nodes in our overlay, thus allowing dynamic MANETs to connect to WANs through reach-back links. Again, it is important to notice that while DENS is a proprietary research protocol, we leverage the open Web services standards, maintaining compatibility with existing clients and services.

[7] argue that the Publish/Subscribe paradigm can be used effectively to facilitate coordination of mobile users, for example, in a disaster recovery application: Rescuers equipped with networked devices (e.g., PDAs) can publish information, and other members can selectively subscribe to the information they need to perform their tasks. The authors argue that scalability is an essential condition of the Publish/Subscribe system, and propose a Publish/Subscribe system suitable for large MANETs. They combine document flooding and content-based routing techniques in a hierarchical manner, and evaluate their solution in a simulator. Our work is similar to this, in that we address and solve the same problem of creating a scalable Publish/Subscribe solution for MANETs. However, in order to stay interoperable with existing software we have based our solution on topic-based routing and open Web service standards.

[5] argue that Web services and Publish/Subscribe-based schemes up until now mostly have been considered separately, and that it is not clear that a possible unification will adhere to any overarching, pre-planned approach. To address this situation they developed a theoretical and conceptual framework extending current Web service programming models and describing the necessary underlying middleware. While the implementation of such middleware was beyond the scope of their paper, an infrastructure based on collaborating brokers was outlined. In this respect, our work is similar to this, in that we employ an overlay network where DSProxy instances take on broker responsibilities. [5] emphasizes the importance of exerting as small an impact as possible on the existing Web services programming models. However, as our middleware solution aims to leverage the power of Publish/Subscribe schemes with existing Web services and clients, it requires no extensions to existing Web service programming models.

PUSMAN [3] is a middleware system for topic-based Publish/Subscribe in MANETs. It uses a collection of brokers to forward advertisements and subscription information. By detecting mobility through monitoring, PUSMAN will reconfigure its overlay to ensure a high delivery success ratio. The work done here is orthogonal to our own; we aim to ensure delivery through the use of a store-and-forward mechanism, which could potentially be improved by combining it with the reconfiguration approach used in PUSMAN.

3 The DSProxy

The DSProxy system [6] is our prototype middleware solution developed in Java. It is a novel, lightweight and cross-platform system with pluggable components. The core DSProxy features include providing store-and-forward capability to SOAP messages, utilizing compression of SOAP and XML and facilitating the traversal of multiple heterogeneous networks. At the same time, it remains compliant with unmodified COTS Web services and clients. By placing one or more DSProxys between a Web service and a Web service consumer, store-and-forward functionality is introduced into the network, which provides increased robustness in dynamic, heterogeneous networks and MANETs. The DSProxy instances self-organize into an overlay network using an internal service discovery mechanism based on UDP multicast (or it can be statically configured where UDP multicast is unavailable). For more details on how the overlay network is built and organized, we refer to [6].
Once organized into an overlay network, DSProxys exchange information about advertised services and the number of hops required to get there. The overlay network allows for smart routing of Web service requests at the application level, but utilizes the underlying routing protocol for IP routing.

3.1 DSProxy Core Features

Figure 1 displays a simple network layout comprising two separate networks and three physical nodes; the client machine, a gateway machine, and the server. The gateway node is equipped with two network adapters providing a physical data link to each network. A standard Web service client and a Web service run on the client machine and the server respectively, and a DSProxy instance is running on the gateway machine, effectively bridging the two different networks together. This enables communication between these two networks even if no IP-level routing is available. Additionally, DSProxy instances run on both the client machine and on the server. When the client wishes to invoke the Web service residing in the other network, instead of initiating an end-to-end connection directly to the server, it sends the SOAP invocation request to its local DSProxy instance, which relays the request to the gateway DSProxy, and so on.

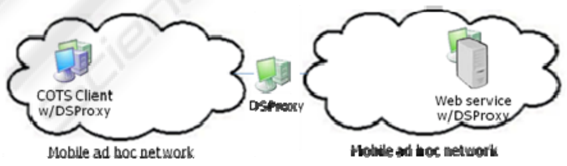

Figure 1: A simple network layout showing two discrete networks bridged with a DSProxy, adding store-and-forward capability to a standard Web service.

The only difference between a direct invocation of a Web service method and an invocation of the same Web service method through the DSProxy overlay network is the URL used to address the service. The original URL is replaced with a URL in the following form:

http://127.0.0.1:7000/?uniqueServiceName=weatherService

The 127.0.0.1-address indicates that the Web service client is relaying the invocation request through a DSProxy instance running on the same physical machine, and the uniqueServiceName-parameter instructs the DSProxy overlay network to route the invocation request to the DSProxy instance responsible for invoking the Web service (typically the DSProxy running on the server hosting the service). The TCP-connection between the Web service client and the first DSProxy is kept open until the DSProxys return the response data.

All three DSProxys are part of the same overlay network, and by monitoring their environments, all DSProxys know their neighboring DSProxys, and where to route a request in order to invoke a particular service. Upon receiving a service invocation request, the gateway DSProxy then relays it to the server DSProxy. Finally, the server DSProxy, knowing it is within reach of the actual Web service, invokes the service, and returns the response data using source routing (back-tracking the invocation route).

While based on ordinary Request/Response-principles, this deployment offers two important benefits: First, store-and-forward capabilities are introduced into the network. This means that if any of the data links become unavailable, the DSProxy closest to the broken link will store the invocation message and retry the transmission at regular intervals. It can also choose another route to the destination if available. If, for instance, the link from the client (e.g., a search-and-rescue agent, part of a MANET) into the network breaks down, the DSProxy instance running locally on the client machine will cache the request, and thereby provide store-and-forward capability from the very first hop and on into the network. Note that running a DSProxy local to the client is not required, but it is usually advantageous, as described above.

Second, the DSProxys eliminate the requirement of initiating an end-to-end connection between the client and the server. In order to stay interoperable and standards-compliant with COTS Web services, HTTP and TCP are utilized between the client and the first DSProxy and between the last DSProxy and the Web service, while any transport protocol may be used between DSProxy instances. The latter can be of great importance when operating in highly dynamic MANETs, where changing topologies and unreliable data links may require other protocols than connection-oriented TCP.

As long as DSProxy instances are running locally on both the client machine and the server, TCP-connections can always be used for the first and last hop (intra-machine), no matter what kind of networks and data links that connect them.

3.2 DSProxy and Publish/Subscribe

The first development iteration of the DSProxy focused on enabling heterogeneous network traversal and bringing robustness to Web services through store-and-forward capabilities. In the second iteration we focus on bringing together standard Web services and the Publish/Subscribe paradigm.

Figure 2 illustrates how, by utilizing two DSProxys, one can enable Publish/Subscribe operation with COTS Web service clients and services. By deploying a DSProxy instance on the same physical machine as the Web service, the DSProxy can perform frequent, continuous polling of the Web service, without adding any traffic load to the network. This intra-machine polling enables the DSProxy

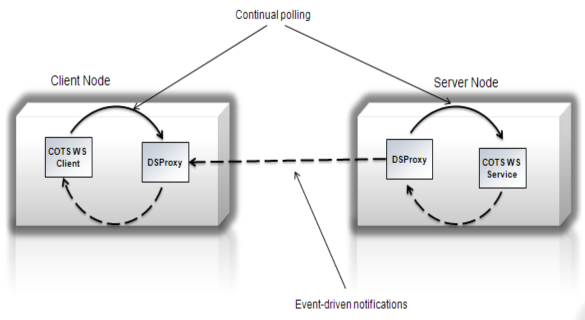

Figure 2: Two DSProxy instances enabling a COTS Web service and client to perform in a Publish/Subscribe fashion.

to discover updated information almost instantly, and the DSProxy may then notify anyone interested in the information, in this case, the DSProxy instance running on the client machine. The server DSProxy will create a hash over the Web service response data, and compare it to the previous response data hash, to determine whether the response is identical (and thereby already been pushed to subscribers) or not.

On the client machine, the same approach is used; the Web service client is instructed to poll the DSProxy instance (sending normal Web service requests) running on the same physical machine continuously. This circumvents the inability of Web service clients to accept incoming connections, and enables the client to receive information almost instantly by polling the DSProxy frequently. By utilizing this polling mechanism on both the client machine and on the server, only the machines' internal buses are burdened, and no additional traffic enters the network. Only when the server DSProxy discovers new, updated information is the network utilized, and the information is pushed from the server DSProxy to the client DSProxy, with the former initiating a connection to the latter. These mechanisms enable regular Web services and clients developed with COTS Web service software to communicate in a Publish/Subscribe-based fashion, and will be referred to as the DSProxy native Publish/Subscribe scheme. The practical minimum Publish/Subscribe-enabling setup requires only two instances, preferably deployed on the client and server machines to achieve intra-machine polling. However, it will often be beneficial to deploy multiple DSProxy instances into a network, as this will increase robustness through multiple store-and-forward points and alternative routes between client and service. As with all inter-DSProxy communication, any transport protocol may be used for sending the notifications, depending on the underlying network class and characteristics.

The server DSProxy can be configured to poll different Web services at different intervals, depending on the nature of the service, the latency requirements, and the resource constraints. For an instant messaging service, one would typically require low latencies and frequent polling, and the DSProxy might poll the Web service every second, in order to facilitate fast message exchange. For a weather forecasting service on the other hand, the latency requirements may be substantially lower, and a polling frequency of 5 minutes might suffice.

3.3 DSProxy and Subscriptions

In order to subscribe to a Request/Response-based Web service, the client needs to inform the overlay network about this. If we compare the subscription request to a regular invocation of the same Web service, the subscription is set up simply by doing a small modification of the URL used for the regular service invocation.

Using the example from Section 3.1, assume that the Web service client wants to subscribe to the weather service, i.e., receive new forecasts as they become available, instead of having to poll the service at regular intervals using Request/Response. Although the service itself is unaware of the Publish/Subscribe concept, the DSProxys allow the client to set up a subscription by just slightly modifying the URL described in Section 3.1:
http://127.0.0.1:7000/?uniqueServiceName=weatherService&pubSub=true
Here, the extra parameter pubSub=true instructs the DSProxy overlay network to handle this as a DSProxy native Publish/Subscribe subscription. The first DSProxy will then determine whether the requesting client is already subscribing to the Web service method. If not, it will create a subscription and store it locally. Next, it will forward the request to the next DSProxy instance in the network (one hop closer to the service), and this DSProxy will perform the same check. If this DSProxy is the one responsible for invoking the actual service, it will do so, and start the server DSProxy polling cycle explained earlier. This polling will continue as long as one or more subscribers are active, meaning they have subscribed to, and not yet unsubscribed from, the service.

When a Web service client wishes to unsubscribe from the service, it simply replaces the pubSub=true-parameter with the pubSub=unsubscribe-parameter, which causes all involved DSProxys to delete the subscription. On the server DSProxy, if the unsubscribe-request results in no subscribers any longer being active, the polling cycle for the specified Web service method ends.

The DSProxy native Publish/Subscribe mechanism enables Publish/Subscribe to be used with standard, COTS Web services and clients. However, as implementations of WS-Notification and WS-Eventing continue to mature, and become more plentiful and widespread, we anticipate benefits of being able to interoperate with such clients and services as well. Therefore, when using a WS-Eventing client together with the DSProxy overlay network, it becomes part of the Publish/Subscribe-tree by sending a WS-Eventing compliant subscription message using a URL on the following form:
http://127.0.0.1/?uniqueServiceName=weatherService&pubSub=wseSubscribe
By giving the pubSub-parameter the value wseSubscribe, the DSProxy overlay network is instructed to interpret the subscription message accordingly. This will create one subscription that covers all requests for the same combination of service, filter dialect and filter expression (i.e., a new request for the same combination will not be forwarded, since the subscription is already established). Before forwarding the re-

quest to the next DSProxy in the network, the sending proxy will modify the WS-Eventing subscription message. Specifically, it alters fields such as NotifyTo, to ensure the actual WS-Eventing service sends its notification through the DSProxy overlay network (addressing the last DSProxy instance in the chain).
While Figure 2 presents a simple, conceptual Publish/Subscribe network layout, larger, more complex Publish/Subscribe-trees can be created. Figure 3 shows such a tree, presenting a hierarchical Publish/Subscribe structure. As shown, subscribers to a DSProxy can be both Web service clients and other DSProxys. Such a tree optimizes the flow of information, and can reduce bandwidth requirements in situations where many clients are interested in the same information. When DSProxy C retrieves updated information from the Web service (by polling), it will actively notify DSProxy A and B, and make the same information available for Client 4, allowing it to retrieve the information during its next polling cycle. Subsequently, DSProxys A and B will make the information available for their clients, ensuring that all subscribing clients eventually retrieve the new information. DSProxy C will not delete its subscription to the Web service and end the polling cycle until DSProxy A, DSProxy B and Client 4 have all unsubscribed to the service.

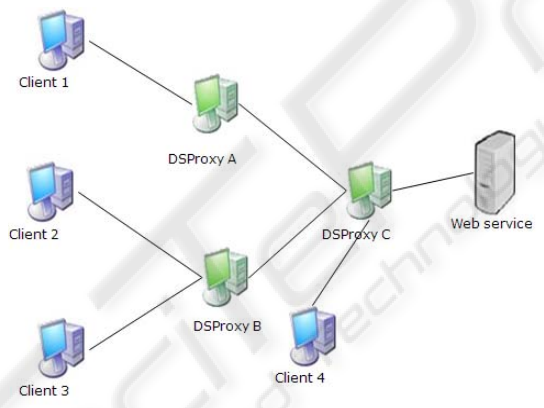

Figure 3: A Publish/Subscribe-tree comprising three DSProxys in a dynamic hierarchical structure.

In highly dynamic MANETs with frequently changing topologies, such a Publish/Subscribe-tree is susceptible to failures, as nodes re-arrange, become unavailable, and disappear without warning. However, the DSProxy Publish/Subscribe-mechanisms are built on top of the overlay network, which is self-organizing, and able to adjust to changing environments. When a DSProxy notices that it has not received any notifications within a configurable period of time, it will resend the original subscription-request to the DSProxy now reporting to be the one closest to the Web service. This may or may not be the same DSProxy that it originally subscribed to. If it is not, the aforementioned chain of events will start again, ending in an active subscription and an initiated polling cycle at the DSProxy now closest to the actual Web service. If it is still the same DSProxy, it may simply be the case that no new notifications have been produced, and it will merely ensure that the subscription is still active.

4 Results and Discussion

In order to verify the DSProxy functionality and capabilities, the solution was tested in a series of field trials, with promising results. The DSProxy middleware solution was used for providing store-and-forward capability within a MANET, as well as for bridging it with a separate, static network. As shown in Figure 4, the MANET comprised 3 mobile nodes (deployed in vehicles) and a stationary gateway node, all running IP-based radios. The gateway ran two radios of different types back-to-back, offering physical data links to each network (one radio link effectively functioning as a reach-back link to the static network). Running DSProxys, the static network operators were able to subscribe to services offered by the mobile nodes located in the MANET, such as imaging services and position services. The static network operators would continuously have updated GPS-positions pushed to them from the vehicles, allowing them to track and visualize the positions of the vehicles on a map as they moved.

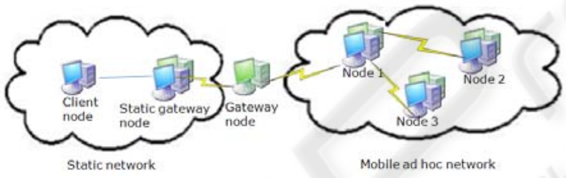

Figure 4: The field trial setup, showing a gateway node running a DSProxy, bridging the MANET and the static network.

When the vehicle operators spotted interesting events, they would take pictures of the events and publish these onto the network. The operators on the static network would then be notified and receive the pictures almost instantly. Given this configuration, with the information having to traverse two bridged, heterogeneous networks and 4 nodes, we experienced typical latencies of 3-5 seconds, from the time that the information was published to it was displayed on the client side. Due to frequent disruption of the radio links, the store-and-forward capability of the DSProxys was demonstrated: As DSProxys running in the MANET attempted to push images to the DSProxy running in the static network, a disruption of a radio link would cause the DSProxys to cache the notifications locally. When the link became available again, the notifications would be re-sent, successfully delivering the pictures to the static network and hiding the erroneous events from the clients and the end users.

This Publish/Subscribe-based interoperation was made possible by utilizing the DSProxy system, and enabled the static network nodes to have information pushed from our regular, non-Publish/Subscribe Web services. It should be noted that such interoperation using the DSProxy native Publish/Subscribe mechanisms does require small modifications to be made to the otherwise regular Web service clients. As with all communication going through the DSProxy overlay network, the URL to the end Web service must be modified, as described in Section 3.1 and 3.3. This information is usually embedded in human readable configuration files (e.g., in locally cached WSDL-files), and do not require recompiling the client application.

In addition, if DSProxy native Publish/Subscribe is to be used, the client must be modified to engage in the polling cycle described earlier. This typically involves wrapping the request-statement within the code in a loop structure. Finally, clients must handle null-responses, which occur when the client-side DSProxy being polled do not offer any new notifications. The two latter modifications are usually quick and easy to implement and do not include touching the actual business logic, but they do require the client to be recompiled.

The Web services on the other hand, require no modifications. Still, considerations should be made when selecting which Web services to Publish/Subscribe-enable through the DSProxy overlay network. As a DSProxy determines whether or not the information is updated based on the hash produced, constantly changing information will produce ever changing hashes, thus, also a constant flow of notifications. For instance, responses that contain fine-grained time-stamps would always produce new notifications, even though the information may otherwise be unchanged.

Although the intra-machine polling mechanism utilized by DSProxys to retrieve updated information from COTS Web services does not generate any network traffic, it does require server resources such as CPU and memory. While a typical Web service invocation only requires a small amount of resources, it may ultimately limit the number of Web service methods and the polling frequency that can be used. In order to establish this limit, an experiment was conducted using a setup similar to the one presented in Figure 1, with one client machine and one server machine, both running one DSProxy instance. Simple “Hello world”-like services were developed using C# and ASP.NET 3.5, which returned strings consisting of 100 random characters for each invocation. This made the services produce new responses for every invocation, which in turn lead to notifications being produced for every poll, in order to produce a “worst-case” scenario.

The services were deployed inside a Microsoft Internet Information Services (IIS) 5.1 Web Server instance running on the server. The server machine was equipped with an Intel Pentium 4 dual core 2.6 GHz CPU, 1 GB RAM running MS Windows XP Pro SP3. A standard Web service client application capable of invoking the services in the normal Request/Response-fashion were developed using .NET 3.5.

The client application ran on the client machine, and by modifying the Web service invocation URL, the invocation requests were relayed through the two DSProxy instances and delivered to the Web service. The URL was also modified to instruct the DSProxy overlay network to initiate Publish/Subscribe, and the server DSProxy would start one polling-cycle for each service subscribed to (running as parallel threads). Because the client application ran on a separate machine, all CPU load on the server was associated with the server DSProxy repeatedly polling the services and notifying the client DSProxy (in addition to some load associated with the actual Web services and the IIS).

The server DSProxy was configured to poll each service once every second, and the number of services subscribed to could be controlled from the client application.

The CPU utilizations were measured using MS PerfMon, and averages were monitored for 60 seconds during each run. Figure 5 shows the varying average CPU loads when subscribing to 5, 10, 15 and 20 Web services. As seen from the graph, the performance scaled close to linearly, ranging from 0.755 % to 4.469 % CPU utilization. Even when 20 different services were subscribed to and polled every second, creating and sending 20 new notifications every second, less than 5% of the available CPU-cycles were utilized on average.

An extrapolation of this data indicates that the current setup could theoretically handle subscribing to nearly 450 services, polling each of them and producing

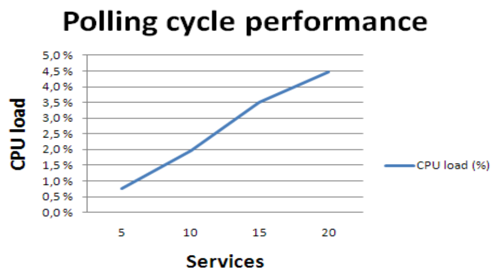

Figure 5: The graph shows CPU utilization for a given number of services being polled every second.

notifications once every second. However, the Web Server Software places limitations on the number of possible concurrent Web service invocations. Also, more complex services requiring CPU-intensive calculations or IO-operations would reduce the performance, dividing the available CPU cycles between tasks associated with Publish/Subscribe and the actual work being performed by the services. On the other hand, most production servers would greatly surpass our test-server performance-wise. In addition, for many services, a considerably lower polling frequency will suffice.

It is important to note that, because the server DSProxy does the actual invocation of the service, the polling frequency is constant, regardless of the number of subscribers per service. This means that in a scenario with many subscribers to a service, the polling frequency may be considerably lower than if each individual client were to poll the service using Request/Response. In fully distributed environments, participants may function as both clients and servers by exposing their own services. The hardware hosting such services may be limited devices such as PDAs, and this should be taken into account when configuring polling cycle frequencies.

While standards-based interoperation with COTS Web services using DSProxy native Publish/Subscribe requires minimal modifications to the Web service clients, using WS-Eventing-based Publish/Subscribe in the DSProxy requires no modifications to any software. By placing DSProxy instances between WS-Eventing-based clients and services in dynamic MANETs, added robustness is achieved through store-and-forward capabilities. Additionally, network performance gains are achieved when multiple WS-Eventing clients are interested in the same information, in other words subscribing to the same services using the same filter dialects and expressions. Instead of the WS-Eventing service and the DSProxys having to send the same information to each of the clients directly, as is the case for regular WS-Eventing and WS-Notification, the information is sent to one or a few subscribing DSProxys. Since the overlay network effectively constitutes a multicast tree, less traffic is relayed through central parts of the network. Client-specific data, such as the

ReferenceProperties-field, are stored and added to the notifications at the DSProxy instances closest to each client.

5 Conclusions and Future Work

Through a series of field trials, the functionality and capabilities of the DSProxy were tested, demonstrating its usefulness in heterogeneous and error-prone networks and showing potential for typical search-and-rescue scenarios. By utilizing the lightweight DSProxy system in both MANETs and static networks, regular Web services can leverage the power offered by the Publish/Subscribe paradigm, requiring only minor modifications to be made to the Web service clients. The practical minimum setup for achieving this would only require two DSProxy instances to be deployed into the network, preferably as close to the client and the service as possible.

Experiments have shown that the DSProxy polling cycles consume relatively low amounts of resources, and together with the fact that the invocation frequency is independent of the number of clients, this means that a server can potentially handle a large number of services and clients. As we expect implementations of WS-Notification and WS-Eventing to mature and become more plentiful and widespread in the future, the DSProxy system supports WS-Eventing, with support for WS-Notification currently under development. The DSProxy WS-Eventing-based Publish/Subscribe mechanisms allow bandwidth-optimized routing of information, requiring no modifications to clients or services. Both modes benefit from the store-and-forward capabilities provided by the DSProxys, facilitating Web service based Publish/Subscribe in MANETs and other unreliable networks. Additionally, both modes of Publish/Subscribe can be used with Web services across multiple heterogeneous networks.

Future work includes compliancy with the WS-Notification standard and additional schemes for optimizing maintenance of Publish/Subscribe-trees. Also, a UDP multicast-based notification capability is under development, which is expected to further reduce traffic loads in radio networks where multiple clients and DSProxys subscribe to the same information.

References

1. Baldoni, R., Beraldi, R., Querzoni, L., and Virgillito, A., 2007, Efficient Publish/Subscribe through a Self-Organizing Broker Overlay and its Application to SIENA, The Computer Journal, volume 50, num. 4, Oxford University Press, pp 444—459.
2. Birman, K., and Joseph, T., 1987, Exploiting virtual synchrony in distributed systems, SIGOPS Oper. Syst. Rev. 21, 5, pp 123-138.
3. Denko, M. K., 2006. Pusman: Publish-subscribe middleware for ad hoc networks, IEEE CCECE/CCGEI, Ottawa.
4. Lekova, A., Skjelsvik, K.S., Plagemann, T., and Goebel, V., 2007, Fuzzy Logic-Based Approximate Event Notification in Sparse MANETs, Proceedings of the 21st International Conference on Advanced Information Networking and Applications Workshops - Volume 02, pp 296-301.

5. Silva-Lepe, I., Ward, M. J., and Curbera, F., 2006, Integrating Web services and Messaging, IEEE International Conference on Web services, Chicago, USA, pp 111-118.
6. Skjervold, E., Hafsøe, T., Johnsen, F.T., and Lund, K., 2009. Delay and Disruption Tolerant Web services for Heterogeneous networks. IEEE MILCOM, Boston, MA, USA.
7. Vinoski, S., 2004, Web services Notifications. *IEEE Internet Computing, vol. 8, no. 2, pp 86-90.*
8. Yooa, S., Sonb, J.H., and Kima, M.H., 2009, A scalable Publish/Subscribe system for large mobile ad hoc networks*, Journal of Systems and Software, Volume 82, Issue 7, pp 1152-1162.*