Interoperable service discovery: Experiments at Combined Endeavor 2009

Frank T. Johnsen
Joakim Flathagen
Trude Hafsøe
Magnus Skjegstad
Nanda Kol (NC3A)

Forsvarets forskningsinstitutt/Norwegian Defence Research Establishment (FFI)

17.11.2009

FFI-rapport 2009/01934

1086

P: ISBN 978-82-464-1724-0
E: ISBN 978-82-464-1725-7

Keywords

Tjenesteorientert arkitektur

Eksperimentering

Nettverksbasert Forsvar

Service discovery

Approved by

Anders Eggen	Project Manager
Eli Winjum	Director of Research
Vidar S. Andersen	Director

English summary

This report summarizes the Web services discovery activities in project 1086. In particular, we discuss our experiments at Combined Endeavor, where we performed joint experiments with the NC3A in the Netherlands in 2009. The standardized Web services discovery mechanisms are well suited for use in networks with high bandwidth and fixed infrastructure, whereas experimental solutions must be used in disadvantaged grids. Through our experiments we show how interoperability between the experimental and the standardized mechanisms can be achieved using service discovery gateways.

Sammendrag

Denne rapporten oppsummerer aktivitetene innen Web services discovery i prosjekt 1086, og tar spesielt for seg eksperimentene vi utførte på Combined Endeavor i samarbeid med NC3A i Nederland i 2009. De standardiserte Web services discovery-mekanismene egner seg godt i nettverk med høy båndbredde og fast infrastruktur, mens eksperimentelle løsninger må benyttes i disadvantaged grids. Vi viser dette i eksperimentene, og tar for oss hvordan interoperabilitet mellom de eksperimentelle mekanismene og de standardiserte mekanismene kan oppnås ved hjelp av service discovery-gatewayer.

Contents

1 Introduction

The NC3A and FFI performed a joint experiment at Combined Endeavor 2009 (CE) in the Netherlands. We interconnected two mobile ad hoc networks (MANETs) and two wired intranets, which served as tactical mobile and tactical deployed networks during the experiments. It should be noted that in this report we use the term "MANET" in a general sense, referring to both military and civil technology supporting wireless, mobile networking in an ad hoc manner. The goal was to demonstrate cross domain Web services, featuring such aspects as service discovery and service invocation. Delay and disruption tolerant proxies (DSProxys) were used to ensure cross domain Web services invocation. Service discovery was achieved by using suitable mechanisms in each network, and gateways for interoperability.

1.1 Combined Endeavor

In previous experiments we have shown that it is possible to invoke Web services *in* military networks. At CE, we wanted to explore the use of Web services technology in a combined operation, by employing service discovery and invocation both *in and across* heterogeneous military networks. We wanted to employ Web services standards as much as possible, augmenting the system with proprietary, experimental solutions only where necessary. We wanted to test our prototype DSProxy, which can enable COTS Web services clients and services to operate across heterogeneous environments. By deploying the proxy software locally in each network node, the proxy can intercept the standard Web services invocation locally. This means that SOAP over HTTP and TCP is used between client and proxy, and between proxy and server. However, the proxy supports compression, multiple transport protocols and adds delay and disruption tolerance, meaning that the communication *between* the proxies can be performed across heterogeneous networks. A simple deployment like this with locally deployed proxies communicating across heterogeneous networks is shown in Figure 1.1.

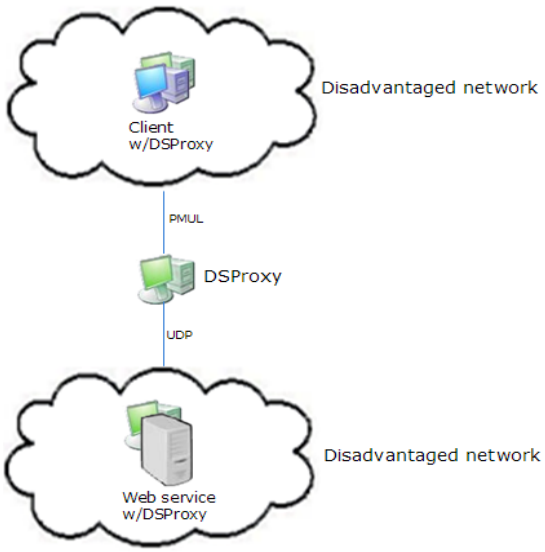

Figure 1.1 Locally deployed proxy configuration.

Also, we wanted to achieve pervasive service discovery. In previous experiments we have performed the service discovery at *design-time* (i.e., the service endpoints used in the experiments have been hardcoded and static in the applications). This way of using Web services is common in civil applications, where the services and the network infrastructure are stable. In tactical networks, however, there is a need for *run-time* discovery, since the dynamic nature of the system means that services are transient. That it is possible to use Web services standards at the strategic level where the infrastructure is based on Internet technologies is obvious. Web services were designed for use in such networks. However, we wanted to experiment with this technology at the tactical level, both for deployed and mobile networks. This report focuses on the service discovery aspects of the experiments.

We wanted to explore two cases:

- First (see Figure 1.2), we wanted to show pervasive use of Web services (i.e., discovery and invocation) across network and national boundaries. We used a traditional setup, where direct communication between the two MANETs was not possible. Instead, all communication had to go via the interoperability point between the two HQs. Interoperability between the nations was provided using TACOMS (communications standard for joint operations, see http://www.tacomspost2000.org/) and the common CE backbone.
- Second (see Figure 1.3), we wanted to try another use case: That of direct interoperability between the two MANETs. This required the use of another interoperability point to connect the two different technologies together.

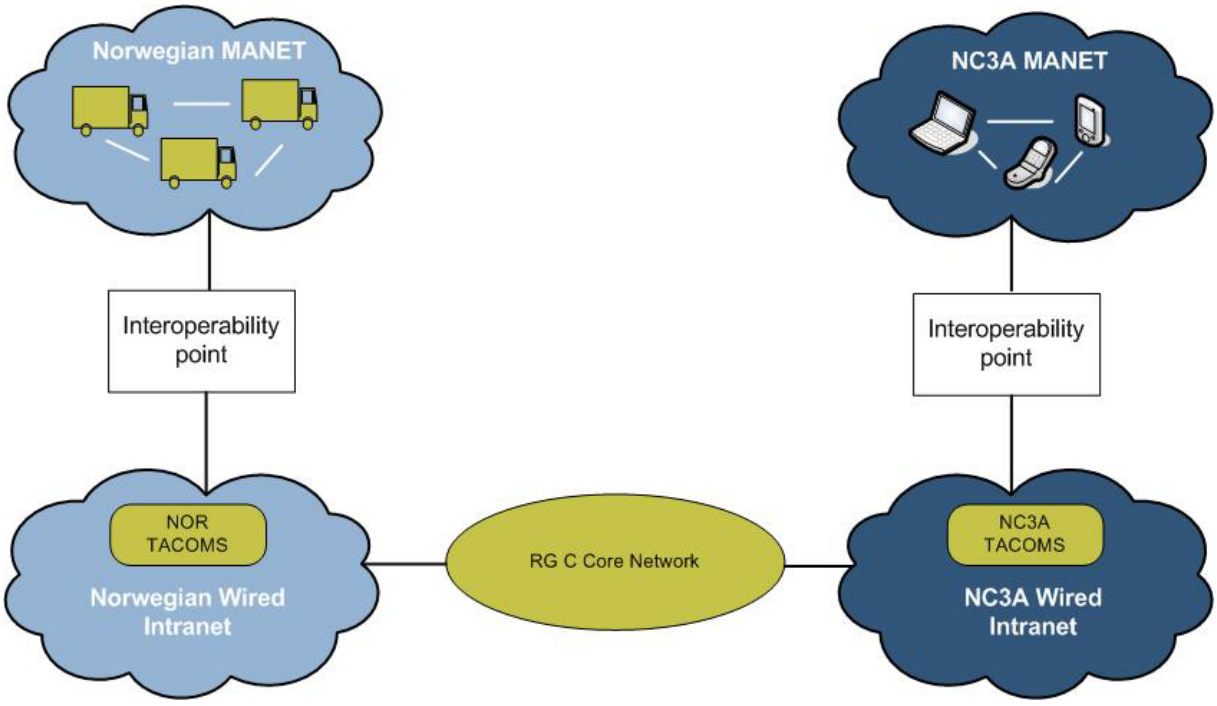

Figure 1.2 CE experiment setup, interoperability between HQs.

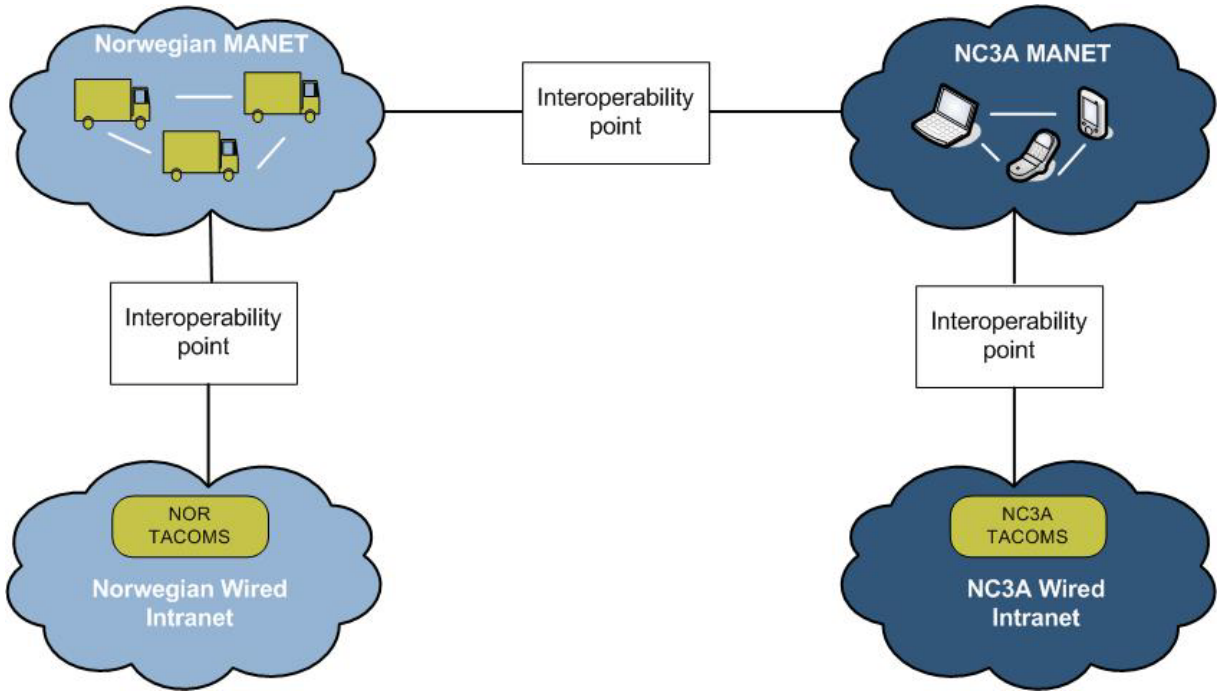

Figure 1.3 CE second experiment setup, interoperability between MANETs.

We provided several services in the networks, and used applications for blue force tracking, chat, etc. For a more complete discussion of the scenario and applications, see [39]. Since blue force tracking is all about getting position information about friendly forces, it is also in a way related to service discovery since it can enable you to find out *where* your services are located in the field, if you know both the services provided by the node and its position. Our experiments used NFFI for blue force tracking.

1.2 NATO Friendly force information (NFFI)

NATO developed this format for blue force tracking in Afghanistan, the NATO Friendly Force Information (NFFI) Afghanistan Force Tracking System. The current version of NFFI is 1.3 as published in draft STANAG 5527. NFFI consists of a message definition and message protocols. The message format is defined by an XML schema containing both mandatory and optional fields. The position data is a mandatory part of the document, and contains information about position (longitude, latitude, altitude) and velocity. Identification data is also a mandatory part, and contains information about the object's name and a 15 character text string from APP-6A/Mil STD 2525B. Thus, the position and identification data contain all the information needed to draw a symbol on a map. Furthermore, a status field contains the operational status of the object. All the other fields are optional, and may contain contact information, telephone numbers, etc. An NFFI track comprises a lot of information (see Figure 1.4). However, most of this information does not need to be sent in every update, since it is either static or has a high classification. Basically, the only part of an NFFI track that needs to be transmitted in the tactical battlefield is the mandatory part of the NFFI information, i.e. the position and dynamics data. This information is defined as a PositionDataType (PDT) in the NFFI schema. The HQ can collect several PDTs and build a complete NFFI message, potentially adding some of the extra, optional information that can be of use higher up in the hierarchy.

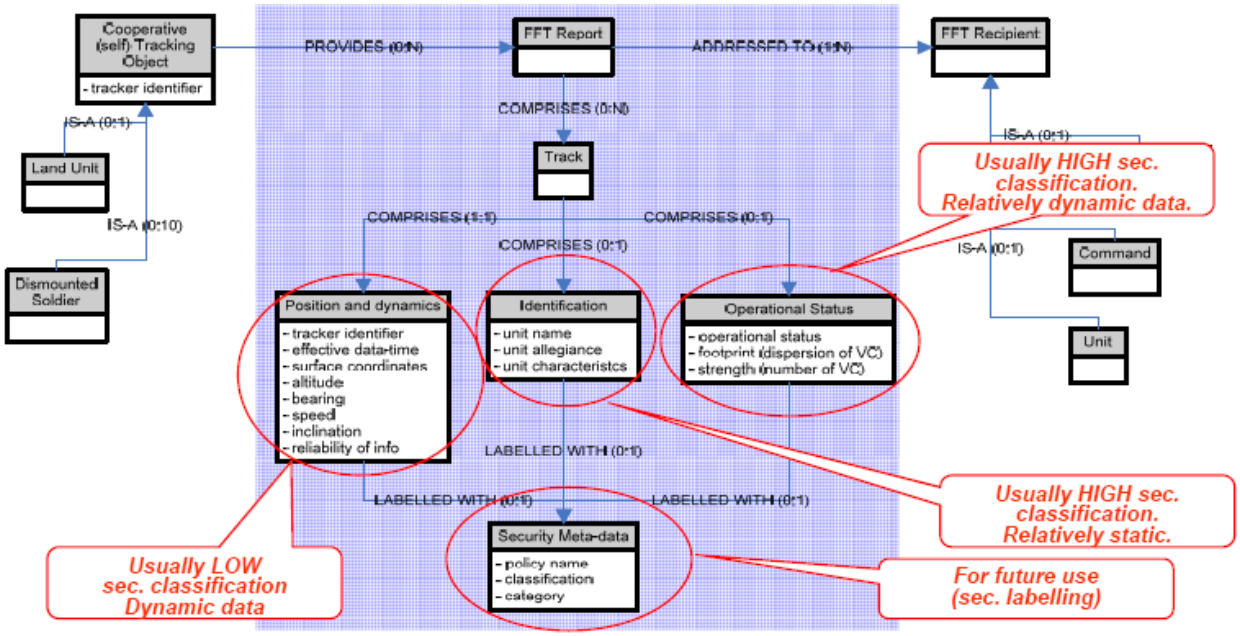

Figure 1.4 NFFI tracks showing mandatory and optional data, along with usual security requirements (from [27]).

2 Motivation - service discovery at (and across) different operational levels

One of the main goals of Network Enabled Capability (NEC) is to increase mission effectiveness by interconnecting military entities. Sharing information between decision-makers can help guide them towards making the right decisions at the right time, and a common information infrastructure is needed to facilitate sharing of relevant information across system and national boundaries. This leads to a requirement for a flexible, adaptable and agile information infrastructure which can support all the information needs of national forces, and at the same time support interoperability. The information infrastructure will have to support a number of different usage scenarios, from fairly static environments where services are stable, to dynamic environments where both services and service users come and go in a non-deterministic fashion. The NATO NEC feasibility study envisions the concept of a Service Oriented Architecture (SOA) to become pervasive in this information infrastructure. In a SOA, networked resources are made available to others as a collection of services, often implemented using a technology called Web Services. Current Web Service solutions are designed for Internet-type networks where bandwidth is abundant and nodes are stationary. Applying such technology directly for military purposes may not be feasible, especially when considering the tactical level where resources are scarce (low bandwidth) and the network consists of mobile units leading to frequent topology changes.

In a highly dynamic environment, being able to locate and invoke Web Services becomes a major challenge. The process of identifying a service, known as service discovery, is an important part of any SOA, but is particularly challenging in dynamic environments such as military tactical

systems. A service discovery architecture for such an environment should reduce the amount of manual configuration, enable automatic discovery and selection of relevant services, and offer a complete and up-to-date picture of the services available at any given point in time. The discovery infrastructure must provide a fresh view of available services. Responses to queries should mirror the current state in the service network and should not advertise services that are no longer present on the network. This is known as liveness information. Moreover, the infrastructure should be robust in terms of partial failure as well as bandwidth efficient, since nodes in dynamic environments may have wireless connections with low network capacity.

2.1 Operational levels

An operational network is complex (see Figure 2.1). There are different levels with different communication needs and solutions. It is apparent that a single service discovery mechanism will not meet all demands. At the lowest level, there are a few highly mobile units. Moving up in the hierarchy there will be more units, but less mobility. The command posts are typically deployed tactical networks.

Looking at the different operational levels in context (see Figure 2.2) we can see that the characteristics vary from level to level. A strategic network has infrastructure and is typically not dynamic.

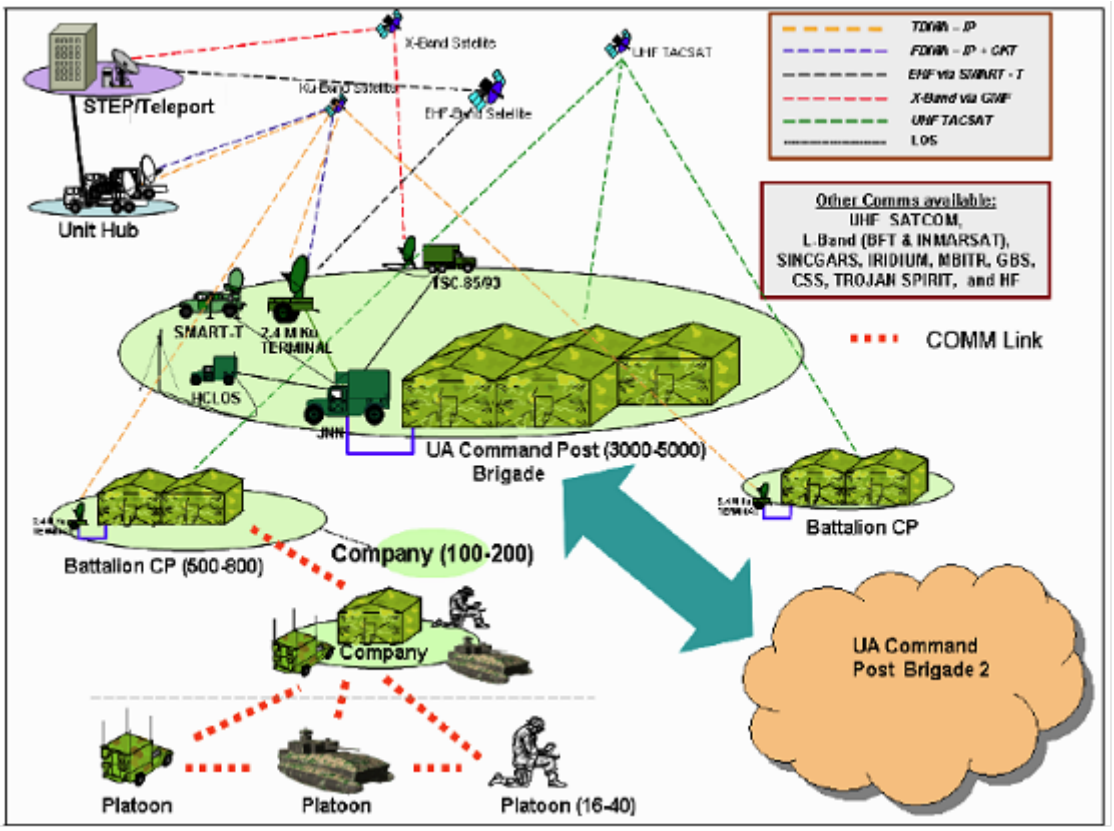

Figure 2.1 Operational network (from [1]).

A tactical deployed network is more dynamic than a strategic network, typically depending on radio or satellite communication. A mobile tactical network, i.e. networked single units taking part in an operation, typically yield the highest dynamicity seen in an operational network. On the other hand, the total number of services available will be highest in strategic networks. The deployed tactical network will have a more specialized need for services, and thus most likely a lower number than on the strategic level. A mobile tactical network will have and use the least number of services. Not only does the limited bandwidth at this level limit the possible amount of services and communication, but also the need for services will be location and mission specific.

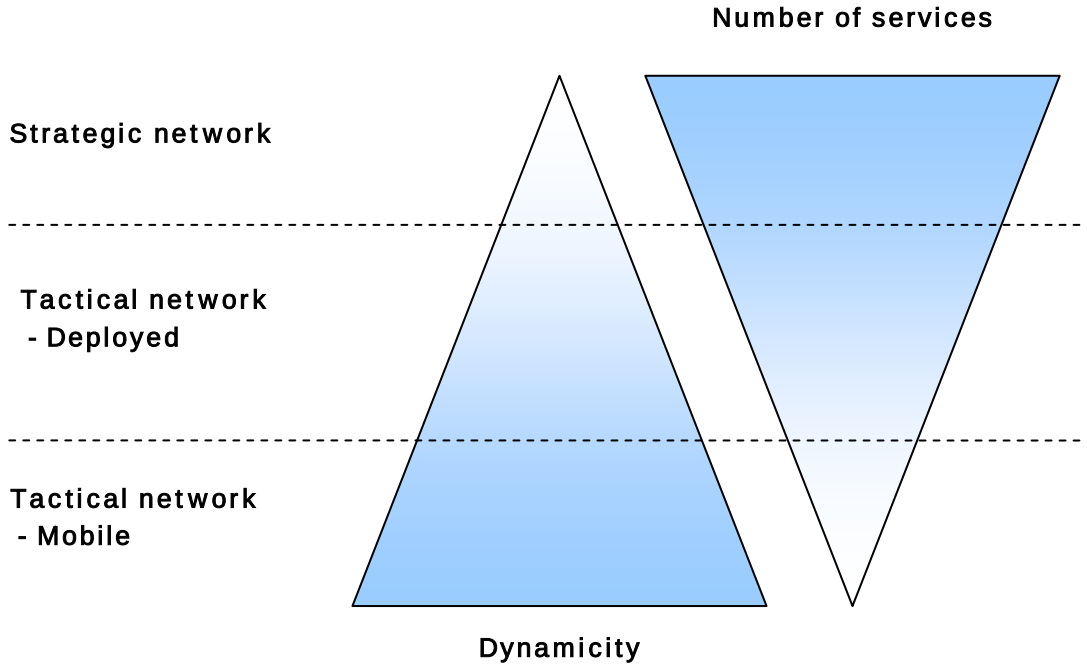

Figure 2.2 Domain complexity.

No single, currently available service discovery mechanism can fulfill all the demands that a military operational network places on service discovery. Because of this, we suggest an architecture where each operational level can use the service discovery mechanism best suited for that network type. For instance, in a network deployment as the one shown in Figure 2.1, a service discovery solution consisting of three different mechanisms could be suitable. Such a solution, illustrated in Figure 2.3, would utilize registries in the strategic network, peer-to-peer (P2P) mechanisms between deployed units, such as command posts, and ad hoc mechanisms between individual units in the field.

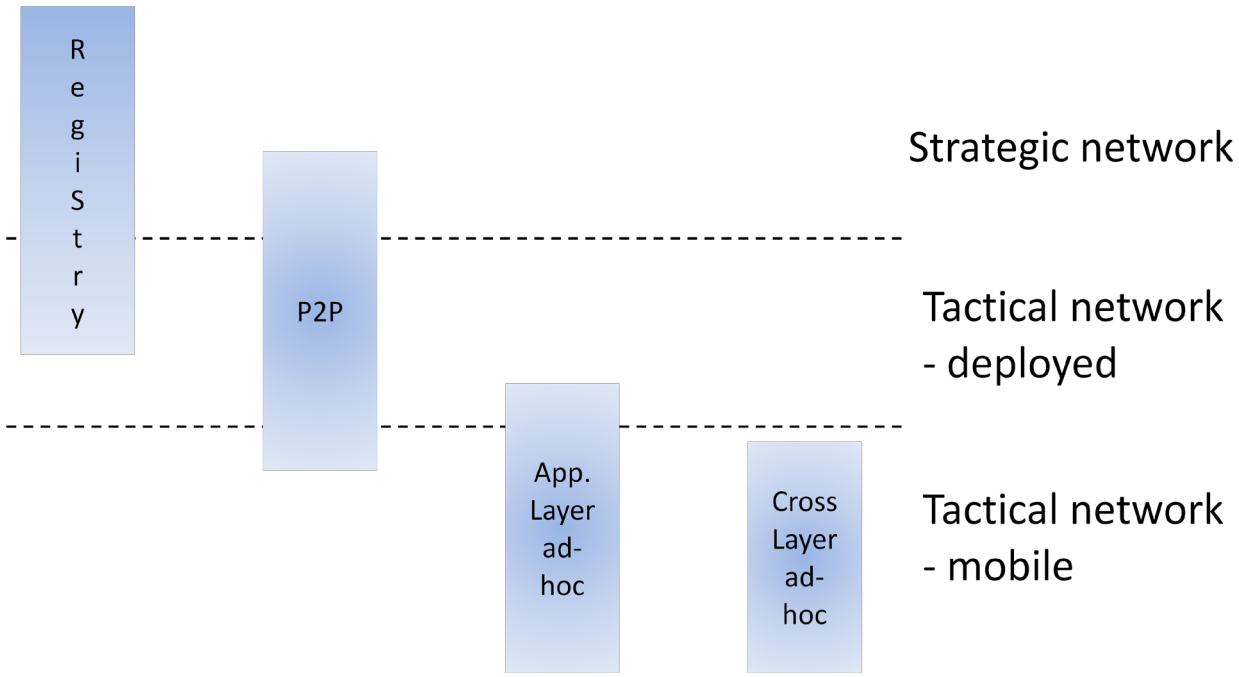

Figure 2.3 Suggested service discovery mechanisms for each operational level.

2.2 Vertical integration

Units in the *ad hoc layer* are typically sensors and services that are mobile. They will primarily need to interact with other services in the same geographical area, but services in other layers should be available on demand. An example of a unit in this layer could be a soldier equipped with simple sensors for localization to keep track of other team members or directions for the current mission. These small ad hoc networks may have limited resources and should use resource friendly discovery methods. Such methods could include *cross-layer discovery*. It is important to note that on the lowest tactical level, e.g. within a squad, bandwidth may be less of a problem because higher bandwidth WLAN technology can be used within a short range. The main problem is often the reach back link, which can have a long range and correspondingly low bandwidth.

Contact with upper layers could be offered as a service published by units equipped with the necessary resources to perform the communication. This gateway service, if present, may be used to send queries to a peer-to-peer layer or directly to a registry if available. Typically a query must be sent to a registry if it contains complicated semantics or in some other way demands more resources than the current node can provide. For example, a handheld device with limited CPU and memory may not be able to process complicated semantics and perform service orchestration. Other queries may be forwarded to the peer-to-peer-network as results in that layer may be returned more quickly, due to possible value added functionality in that layer. Peers in the peer-to-peer layer may also be delegated resource intensive parts of a query or take care of registry communications before returning the result to the originating unit.

Services in the ad hoc layer are published with a minimal set of descriptions, typically just being based on predetermined identifiers or unique names. Searching will mostly consist of broadcasts and/or multicasts with requests for a specific service. More advanced queries are sent to the upper layers through a gateway service.

Above the ad hoc layer is the *peer-to-peer layer*. Typically units in this layer are larger, mobile units that are able to carry more and heavier equipment than the units in the ad hoc layer, thus allowing more resource intensive discovery techniques. Since the amount of resources may vary in each unit, all units in this layer should be able to adapt to the level of available resources – especially bandwidth limitations.

Searching in the peer-to-peer network will depend on resources available in the nodes performing or responding to the search query. Simple semantic descriptions and template based searching should be supported, but need not be used in every search. Due to the overhead of a full semantic search, such metadata should only be used when necessary. Queries originating from units with limited resources in the ad hoc layer will be simple and thus not incur as much overhead as a semantic search. To save bandwidth it should be possible to use previous knowledge gained to complement the queries before the actual search is performed – for instance information about physical location, available resources and so on.

Service descriptions should always contain liveness information to enable caching of search results. This allows peers to quickly return frequently requested service descriptions and limits the bandwidth required for communication with the registries.
Units in the peer-to-peer layer should also be able to participate in the ad hoc layer. This enables the peer-to-peer network to discover local services without the need of a registry. In addition, some peers should be able to act as gateways for the ad hoc layer when possible. Peers providing gateway services enable quick and potentially redundant access to the peer-to-peer network for ad hoc layer nodes.

Units in the *registry layer* are stationary service registries and are typically located in local or central headquarters. Communication is primarily based on land lines and high capacity radio links, thus restricting their mobility. Their high bandwidth allows them to cooperate in federations or make use of replication, making them less vulnerable to attacks. Search queries in the registry layer may have any level of complexity supported by the registry, probably ranging from simple templates to complex semantic searches. Some registries should be available as a peer in the peer-to-peer layer and as a discoverable service in the ad hoc layer. As a peer, the registry may be able to respond to queries without the need for subsequent connections directly to the registry, thus saving bandwidth. In addition, the peer-to-peer network will allow registries to discover each other. As an ad hoc service the registry can be discovered by passing units, for instance enabling the peer-to-peer network to discover unknown registries when one of its peers is geographically close to it.

2.3 Horizontal integration

In coalition forces today, the various nations have to circumvent their proprietary solutions in order to collaborate. That means that even if they are collaborating in the field, their proprietary radio systems cannot communicate with each other. In order to pass information from one nation to the other, the information must be sent up from the tactical to the strategic level in the nation's network, and then passed through a so-called interoperability point (i.e. a gateway) to the other nation's network, before it is sent down again to that nation's tactical level. This must be done due to incompatibilities between the different communication waveforms and current policy. Also, different national crypto modules make interoperability between radios from the same vendor impossible. Thus, interoperability is currently achieved at the strategic level because communication solutions at this level are based on Internet technology. This solution works, but is less efficient than if the various networks could communicate directly. Direct communication should be achievable in the future considering NATO's current focus on waveform standardization, which would allow interoperability points connecting two different national solutions together using a gateway solution with a NATO waveform. Another way to achieve such interoperability is by connecting different radios back to back. Interoperability is an ambition in NEC, and would allow for service discovery between different nations across the same operational level.

2.4 Web services standards

There are several challenges that arise when attempting to perform service discovery in MANETs. Below we summarize some of the most important challenges as identified by [26]:

- Dynamic environments may lead to frequent change in both service metadata (service descriptions) and the topology of the nodes that are part of the system. Frequent topology change means that both service nodes and registry nodes can come and go.
- A proper service discovery architecture for such an environment would reduce the amount of manual configuration, enable automatic discovery and selection of relevant services, and offer a complete and up-to-date picture of the services available at the given point in time.
- Service discovery should work in environments disconnected from the Internet. Moreover, it should be robust in terms of partial failure.
- The system should allow flexible resource utilization, since capacity (memory, CPU, storage) and connectivity can differ from node to node.
- The infrastructure should support different kinds of service description mechanisms, ranging from simple to rich (i.e. semantic) descriptions. Thus, both normal Web services as well as Semantic Web services should be able to use this infrastructure.

Considering these requirements, we look at each of the existing solutions for Web services in turn: Currently, there are three OASIS standards related to service discovery for Web services. The two registry standards, Universal Description, Discovery and Integration (UDDI) and electronic business XML (ebXML), are for use in wide area networks or local area networks. In order to find a service, the client needs to access the registry, search for an appropriate service,

and choose a service from the query reply. However, this solution was developed for business use over the Internet, where connections are fixed; have a high bandwidth, high uptime, and nodes are immobile. In a MANET the network topology is unpredictable, and if the network is partitioned then only clients in the same partition as the registry will be able to discover services, whereas the clients in other partitions will not. This means that a client residing in the same partition as the service it needs may be unable to use it, just because it cannot access the registry to discover it. In contrast, a client that is able to access the registry may discover a service that it is unable to connect to, since the service resides in another network partition. These two aspects (shown in Figure 2.4 and Figure 2.5) are important drawbacks with registries if you attempt to use them in a dynamic environment such as a MANET.

Web Services Dynamic Discovery (WS-Discovery) attempts to remedy the drawbacks of registries. It is designed for use in local area networks and is based on multicasting queries and responses. This means that by using WS-Discovery, you can discover services in your partition and use them. There is no single point of failure in WS-Discovery, and the query responses mirror the current network state. All three discovery mechanisms support simple service descriptions. The registries have third party support for semantics as well. WS-Discovery in its current form has no support for semantic descriptions at all, and is thus not suitable for discovery of services beyond that provided by simple descriptions. WS-Discovery, being based on SOAP over UDP, is quite verbose and thus requires the network to handle large messages. Since we lack a standard that is suitable for use in *disadvantaged grids*,[1] we need to consider experimental solutions.

[1] A *disadvantaged grid* is characterized by *low bandwidth, variable throughput, unreliable connectivity, and energy constraints imposed by the wireless communications grid that links the nodes.*

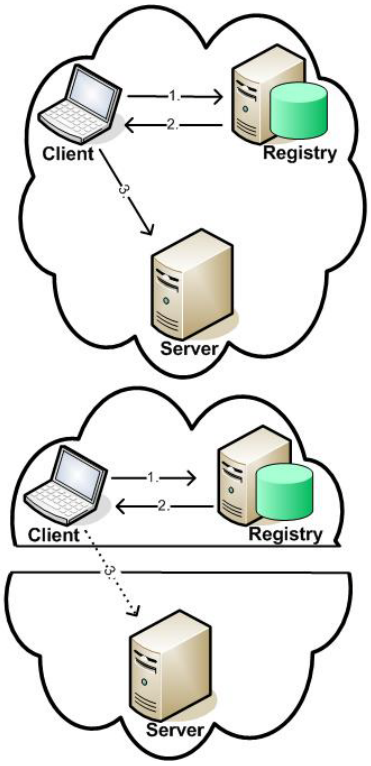

Figure 2.4 Partitioning of the network leaves the server inaccessible.

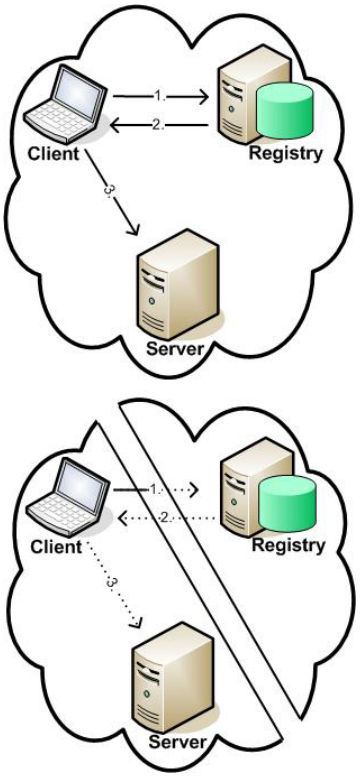

Figure 2.5 Partitioning of the network leaves the registry inaccessible.

3 Evaluation of service discovery mechanisms for tactical mobile networks

Tactical networks can be divided into two types: Deployed tactical networks and mobile tactical networks. The former are typically networks that are moved to some location, deployed, and then being set up and used only when they are stationary. The forward deployed HQ falls in this category. The latter are the MANETs, where movable units participate in a network that needs to function both when the nodes are stationary and when they are moving. Typically, soldiers and vehicles taking part in the network enabled battlefield fall in this category. In this chapter we focus on the tactical networks (both deployed and mobile), and attempt to identify service discovery mechanisms that are suitable for use there.

3.1 An overview of MANETs

A MANET is a collection of mobile nodes connected by wireless links able to dynamically form an autonomous multi-hop radio network—without the use of any pre-existing infrastructure. Intermediate nodes in a MANET can act as routers to forward packets on behalf of other nodes. With their self-forming nature and their ability to cope with rapid changes of the topology, ad-hoc networks are attractive to a variety of applications. However, it is worth noting that ad-hoc networking introduces a great many challenges and imperatives, and also adopts the side effects of wireless computing [4]. Wireless links are significantly less reliable than wired media, they

have unpredictable signal quality and transmission range, the channel can be time-varying and possible asymmetric, and the wireless links are vulnerable to security attacks not found in wired networks. Further, the multi-hop nature in MANETs introduces challenges due to the topology dynamics, heterogeneity, variations of node availability and power constraints. This puts tough requirements to the chosen MANET routing protocol. The IETF MANET working group considers mainly two routing approaches: Reactive routing such as the Ad-hoc On-demand Distance Vector protocol (AODV) [5] and Proactive routing such as Optimized Link State Routing Protocol (OLSR) [6].

3.1.1 Reactive routing

Protocols in this category are reactive in the sense that they attempt to discover routes between nodes only on-demand. Reactive protocols are also named *source initiated* or *on-demand* routing protocols. AODV is one of the most prominent protocols in this category.

3.1.2 Proactive routing

In contrast to reactive routing protocols, proactive routing protocols seek to maintain routes to all nodes regardless of upper layer communication demands. By exchanging control messages periodically, the routing table can be kept updated and fresh routes can be provided immediately.

Compared to reactive routing protocols, this approach yields more control message overhead, but no initial delay to set up a route prior to communication. OLSR is one of the most prominent protocols in this category. For example, it is being used as the routing protocol for the tactical radio WM600 [7].

3.1.3 OLSR

The Optimized Link State Routing Protocol (OLSR) for MANET is a proactive, link-state routing protocol where each node maintains topology information by periodically exchanging link state messages. The novelty of OLSR is to employ multipoint relays (MPRs) to minimize the number of control messages flooding in the network. Each node chooses a subset of its one-hop neighbors as MPRs in such a way that these MPRs will cover all two-hop away neighbors. Hence, messages are forwarded only by MPRs, and not by all nodes (see Figure 3.1).

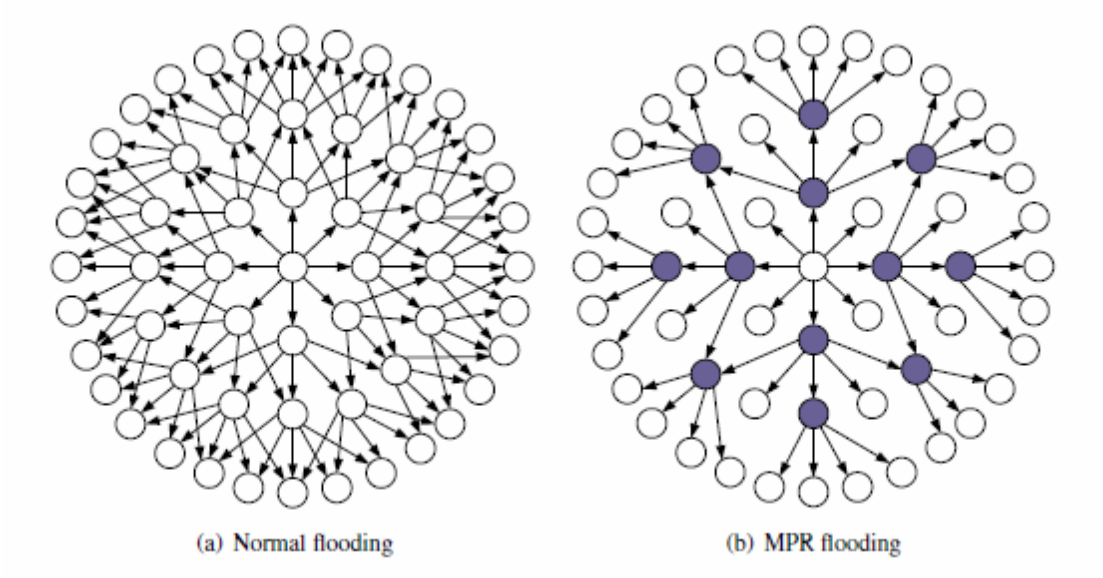

Figure 3.1 Flooding in a multi-hop network. Flooding through multipoint relays (MPRs) reduces the number of duplicate transmissions.

The core functionality of OLSR is: Packet format and forwarding; link sensing with hello messages; neighbor detection; MPR selection and MPR signaling; topology control message diffusion; route table computation; node configuration. Three control messages are defined to provide this functionality:

1. **HELLO** messages are exchanged between neighbors only, and diffuse information about the one-hop neighbors of a node. Upon reception of HELLO messages, the two hop neighborhood is discovered, and further, the MPRs of the given node are chosen. The MPRs chosen by a node is further marked in the following HELLO messages broadcast by that node.
2. **TC - Topology Control**: In OLSR, all nodes chosen as MPRs will transmit TC messages. The TC messages contain the address of the node generating the message, as well as the list of nodes that has chosen the given node as MPR (MPR selectors). TC messages are further flooded using the MPRs, disseminating link state information to all the nodes in the OLSR network.
3. **MID - Multiple Interface Declaration**: The MID message is broadcast by nodes running OLSR on more than one network interface.

In addition, a fourth message type, Host and Network Association (HNA) message disseminates information about OLSR nodes that act as gateways (either to the Internet or to a separate Ethernet).

Using a common header for all message types the OLSR standard provides extensibility of the protocol without breaking backwards compatibility. This feature gives a unique possibility to disseminate additional information through intermediate nodes even if the nodes do not support the specific extension. Leveraging this knowledge made it possible to create an efficient cross-layer discovery mechanism, see the discussion on Mercury below.

3.2 Service discovery mechanisms

There are two main categories of service discovery mechanisms for MANETs:

1. **Application level service discovery**: Refers to protocols *independent of the routing protocol*.
2. **Cross-layer service discovery**: Refers to protocols *integrated with the routing protocol* (be it either reactive or proactive).

Most MANET service discovery proposals belong to the first category, and place service discovery at a layer above routing (Figure 3.2 (a)). Such mechanisms create an overlay on top of the network layer to disseminate service advertisements, requests and replies in the network.

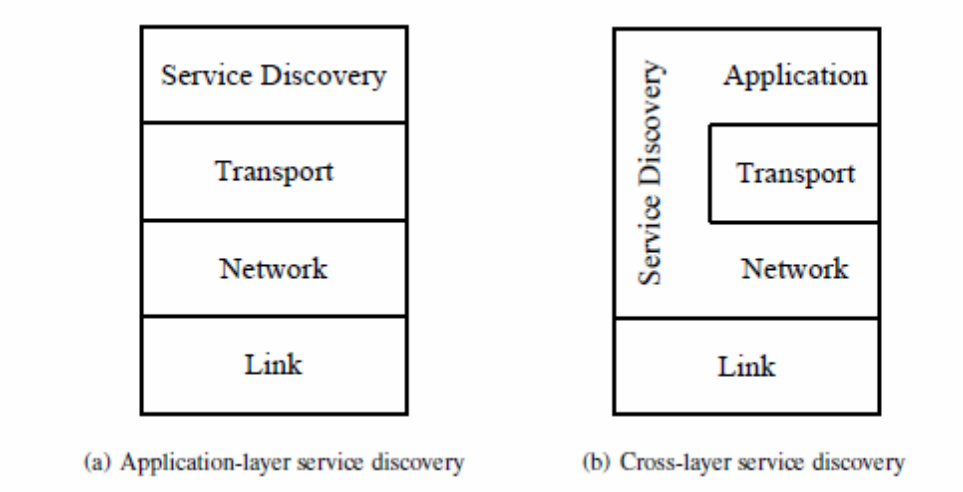

Figure 3.2 Service discovery solutions for MANETs work either at the application layer or in a cross-layer fashion.

There are several advantages using this method: (i) As no assumption is made about the underlying network, it is possible to create pervasive service discovery architecture across different networks domains. (ii) The architecture can be based on existing standards, since the size of the service descriptors is not limited by the routing protocol. (iii) A modular and layered approach is maintained making it possible to replace protocols at any layer.

Cross-layer service discovery is motivated by the need to optimize control overhead and reduce the latency associated with finding services. As both the service process and the routing process must coexist in an ad-hoc network—both processes generate and receive messages. It is therefore possible to exploit the routing layer for efficient dissemination of service control messages (Figure 3.2 (b)).

3.2.1 WS-Discovery

WS-Discovery [22] recently became a standard from the Organization for the Advancement of Structured Information Standards (OASIS) after having been a proposed draft for several years. WS-Discovery addresses the need for a service discovery mechanism for Web services that can

reflect the current state of the network in the absence of a registry, but that can co-exist with and utilize a registry if it is present.

WS-Discovery is a fully decentralized solution which defines a multicast protocol using SOAP over UDP [24] to locate services, a Web Services Description Language (WSDL) document[2] providing an interface for service discovery, and XML schemas for discovery messages. It allows dynamic discovery of services in ad-hoc and managed networks, and enables discovery of services by type and within scope. WS-Discovery leverages other Web service specifications for secure, reliable message delivery. Inherently scalability is limited due to the use of multicast, but WS-Discovery can scale to a large number of endpoints by defining a multicast suppression behavior (i.e., switching from multicast to unicast communication) if a discovery proxy is available in the network. The discovery proxy is intended to be a registry for Web services (e.g. UDDI). When the discovery proxy is discovered, clients use a registry specific protocol to communicate with it. Thus, WS-Discovery can be used to discover registries in a LAN, or services in a local area if no registry is present.

3.2.1.1 Implementation

Since WS-Discovery became a standard during the summer of 2009, there were no standard compliant implementations available yet that we could evaluate. Thus, we chose to look at an implementation of the draft specification [23] instead. We used a Java implementation of the mandatory parts of the WS-Discovery draft, which is freely available from http://code.google.com/p/java-ws-discovery/. Leading up to the CE experiments we were corresponding with the author of this solution, supplying several bug reports. These bugs were fixed, and the version we used at CE is the version later published as “v0.2.0 October 25th”.

3.2.1.2 Discussion

WS-Discovery is not suitable for Internet-scale discovery since it relies on multicast. It is designed for use in local area networks only. That means that you can use it to discover services and registries in a LAN, but not in a WAN (unless you have a discovery proxy in your LAN which is used by WS-Discovery and that registry contains services on the WAN). WS-Discovery, being based on SOAP over UDP, is quite verbose and thus requires the network to handle large messages. Previously we have attempted to use WS-Discovery in a disadvantaged grid, and found that it was not optimal for use there [25]. Therefore we have been researching alternative solutions such as Search+, Mercury, and Service Advertisements in MANETs (SAM) as solutions that can be used in such networks instead. However, considering that there are some minor changes in the WS-Discovery specifications from the 2005 draft version to the 2009 standard, we plan to re-evaluate WS-Discovery in a new series of experiments as soon as a reference implementation becomes available.

[2] For an example WSDL document, and a discussion of its contents, please refer to Section 4.1.1.1.

3.2.2 Peer-to-peer based service discovery

We have experimented with a novel service discovery approach that we have named *Search+* which uses peer-to-peer techniques. The solution is designed to conserve scarce network resources in tactical networks.

The term *peer-to-peer* (P2P) is used to describe networks in which all nodes are treated equally. All nodes are potentially capable of providing services and are also involved in data forwarding. Commonly, a P2P network is created by forming what is usually called a P2P overlay on top of an existing network. The P2P overlay network defines addressing and routing mechanisms and enables nodes (peers) to exchange messages while hiding details of the underlying networks. When a node within the P2P network fails, messages among the remaining nodes can still be exchanged and services provided by the remaining nodes are still available. Thus, a P2P network is highly robust, which is a desired feature in military systems.
Service discovery mechanisms that are currently available for SOAs are not ideal for use in tactical environments. Existing solutions are based on a central registry being available for providers and consumers, introducing a single point of failure. Thus, instead of (or as a supplement to) a central registry a P2P based discovery mechanism would be a better choice. We therefore propose to use a P2P based system to implement service discovery for SOAs in military networks.

In a P2P based SOA each node can fulfill the role of producer, consumer and registry simultaneously. A node can provide a specific service and be a client of services as well. Each node maintains a local registry which contains information about the services provided by the node. However, in most P2P networks the registry content of a node will grow over time as the node learns about services that neighboring nodes provide. The exact learning procedure depends on the P2P algorithms used.

If a consumer requires a specific service it sends a search request through the P2P overlay network. Each node receiving this search request compares it with the information in its local registry and sends a response to the requester when a match is found. After processing a search request a node forwards this request to neighboring nodes. The exact forwarding behavior depends on the chosen P2P overlay.

As no central registry exists, the single point of failure regarding service discovery is eliminated in a P2P based SOA. Different P2P network types exist and it has to be decided which type is suited to implement a P2P based SOA. In general, P2P networks can be classified as unstructured or structured. The two types differ in the way the local registry on nodes is populated and how search requests are routed in the overlay network.

In a structured P2P network specific nodes are used to store specific service information. A node offering a service informs a dedicated node for this service type that the service is available. Hence, each node knows where to direct a search request when looking for a particular service type, which improves search accuracy and speed. The P2P network implements repair strategies

for situations in which a node holding particular registry information becomes unavailable. However, we argue that structured P2P networks are not suitable for a military SOA as the structured storage of registry information again introduces single points of failure. In fact, such systems are similar to a federation of central registries as they are used in non-P2P SOAs.
In an unstructured P2P network nodes send search requests through the overlay network. A node that has a match in its local registry will send a response. In the worst-case situation a search request has to be broadcast to all nodes in the network before a match is found. However, nodes can learn about services offered by neighboring nodes through observation of search response messages. Hence, in a practical deployment worst-case search efforts are rarely required. In addition, the unstructured network has no single failure point.

As the motivation to use P2P techniques in SOA is to increase reliability and to remove single points of failure, an unstructured P2P network should be used. The most prominent example of a protocol for unstructured P2P networks is Gnutella [14].

3.2.2.1 Search algorithms for P2P networks

In this section search algorithms for unstructured P2P networks are described and discussed. The search algorithm is necessary to implement the registry functionality of the P2P based SOA. First, we describe the simple but commonly used flooding search algorithm. We show that such a simple algorithm is not useful for military networks as it is not designed to conserve bandwidth. Second, we describe an existing search algorithm named Advertisement-based Search Algorithm for Unstructured P2P Systems (ASAP) [15], which was designed to reduce bandwidth consumption. Third, we describe our own search algorithm named Search+ which is inspired by ASAP. Search+ is designed to further reduce the bandwidth consumption of the service discovery process. The described flooding and ASAP algorithms are used in the evaluation for comparison with our proposed Search+ algorithm.

3.2.2.2 Flooding

The flooding algorithm can be considered the most basic search algorithm in P2P overlays. When a node performs a search it will send the search criteria to each of its neighbors together with a *time-to-live* (TTL) value. The neighbors will decrease the TTL value and forward the message to their neighbors, continuing this process until the TTL value reaches 0. Each node that has a service that matches the query will send a reply back to the node that performed the search.

The flooding algorithm is simple to implement and has low processing overhead. However, there is a chance that a service is not found even if present. If a node providing the service cannot be reached due to set TTL value a service discovery might fail. In addition, flooding consumes a large amount of bandwidth which is not desirable in a tactical network. Obviously, the TTL value can be used to balance search accuracy (discovery success rate) and bandwidth consumption. However, even TTL restricted flooding becomes too bandwidth intensive if a reasonable accuracy is to be achieved.

3.2.2.3 ASAP

ASAP is an algorithm specifically developed to reduce bandwidth consumption and improve search accuracy in unstructured P2P networks. The algorithm is proactive, meaning that instead of waiting for search queries to arrive and then respond to them, the nodes in the P2P network will actively exchange indexes with information relevant to their interests. The basis for the ASAP algorithm is four observations made by the authors about file-sharing P2P-networks on the Internet:

1. Large parts of the traffic in file-sharing P2P networks consist of search queries.
2. The rate at which queries are performed tends to fluctuate with usage patterns.
3. Content shared on many nodes does not change very often, if ever. Also, downloaders (i.e. clients) do not usually further share the documents downloaded from other peers.
4. Interest clustering is common in file-sharing P2P systems.

We argue that these observations also hold for service discovery in tactical networks. The first point is generally valid for unstructured P2P networks, be it in a tactical network or elsewhere. The second point would correspond with the need to find and use services, and this need would obviously not be constant, but rather fluctuate with the needs of the execution of the current military operation. The third point is also valid, because a service being provided would most likely continue to be available unless something were to happen to the node (e.g. incapacitation or loss of network connectivity), and services would only be consumed by the clients and not shared (i.e. re-published) by them due to the nature of Web services, where nodes are either a consumer or a producer but usually not both. Finally, the fourth point is a trait that can also be anticipated in tactical networks, where different units have different roles in an operation, and depending on the role, the unit will need to consume different information services. Thus, in an operation, units in an area having similar needs will need to access the same set of services, leading to interest clustering.

When a node looks up a service, the node will first inspect its local registry. The local registry contains a list of service descriptions and node addresses indicating where specific services are located. Each service description is stored as a Bloom filter [8] which is a dense representation of a service (a set of bits representing hash values of the service descriptor). When a match is found, the node will know with a certain probability where the service is located. Since this probability never reaches 100%, a message asking for confirmation needs to be sent directly to the node providing the service. If the node acknowledges the match, the search is considered successful. If no match is found, the node will advertise its need for a registry update to all its neighbors. Over time, a node builds up a local registry that is able to answer a nodes service discovery requests.

The nodes in the overlay synchronize their registries by sending registry updates whenever their own registry, i.e. the Bloom filters change. These updates are generally very small, consisting only of a few bytes per filter. ASAP has no single point of failure as registry content is propagated through the network. In addition, it has low bandwidth requirements. This suggests that ASAP is suitable for use in tactical networks. We have implemented ASAP within a P2P network and have performed numerous tests which show that ASAP is indeed very bandwidth

efficient compared to basic solutions such as flooding. However, there is still room for further improvement of the search technique, and in the following section we introduce our own Search+ algorithm which can achieve a bandwidth to hit ratio that is even better than that of ASAP.

3.2.2.4 Search+

Search+ follows many of the same principles as ASAP, in that it uses advertisements with Bloom filters and a similar search and confirmation mechanism. The idea in Search+ is that each node will subscribe to advertisements that match their interests.
Advertisements in Search+ contain a Bloom filter, a node identifier and a version number. This is identical to the full advertisements used in ASAP, and contains the following fields:

- A Node ID, which is a unique identifier for the node that created the advertisement.
- The Bloom filter, which contains a bit pattern used to match keywords against the content held by the producer.
- A list of Topics covered by the content in the Bloom filter.
- A Version number that is incremented for each change to the topics or Bloom filter fields.

Join: When a new node joins the network, its task is to inform the neighboring nodes of the topics (types of services) the new node is generally interested in. To perform the Join operation a subscription message is sent. The TTL value in this message determines how far into the network the subscription request is forwarded. Note that Search+ does not send service advertisements at this point.

Maintenance: The maintenance process in Search+ has two main functions; to process newly arrived subscription requests and to publish service advertisements. Each new request is processed and the specified interest is stored in a table. Each node will remember the interests specified by each of its neighboring nodes.

Note that new subscription request messages are only sent when nodes join or when a node's interests change. The request message always contains the topics (service types) the node itself is interested in and the interests specified by its neighbors. When a node receives a subscription request message with TTL greater than 0 it adds its own interests and its other neighbors' interests to the message before forwarding it.
New advertisements are published by stepping through the list of neighbors and checking for cached advertisements matching one or more of their interests. After retrieving all the relevant advertisements for node N, those that have already been sent are removed from the set. The result is then sent to N and subsequently added to the list of advertisements that N has received.

Search: In the search process, the local advertisement cache is first processed for Bloom filters matching the search terms (the service description). For each match that is found a confirmation request is sent to the node possibly hosting the service. If this fails the search ends as there is no fallback method that could provide more search results. The only option in such case would be to send a subscription message with a higher TTL. However, such measures should only be taken if

deemed absolutely necessary as this action has the potential to trigger substantial amounts of traffic.

Note that this behavior distinguishes Search+ from ASAP which requests new advertisements from neighbors as a fall back mechanism. In the ASAP algorithm this action gradually transports advertisements towards the interested nodes. In the Search+ algorithm however, this process takes place when advertisement subscriptions are sent during the join process.

3.2.2.5 Implementation and evaluation

To evaluate the proposed Search+ protocol we implemented a P2P overlay. We used the Juno [16] framework to implement the software for the P2P nodes. Juno is a reconfigurable middleware for heterogeneous content networking which simplifies the implementation of P2P overlay networks.

As P2P network protocol we selected Gnutella [14]. The Gnutella protocol defines a basic set of message types that can be used to create and maintain a P2P overlay. Standard Gnutella implements a simple flooding algorithm as search algorithm. We also implemented ASAP and Search+ as alternative search algorithms for our Gnutella P2P overlay. Thus, we are able to compare the performance of the three search algorithms: flooding, ASAP and Search+.
In particular, we were interested in seeing if Search+ could improve the search hit rate while reducing bandwidth consumption.

3.2.2.6 Evaluation setup

For evaluation we used a P2P network with 100 nodes offering 400 unique services. In the following paragraphs we describe in detail the network configuration. For further details about the experiment, see [20].

Services and Topics: Each node is sharing a number of services identified by a service name. For the experiment we generated 400 unique services grouped into 14 topics. Each service is associated with one topic; services are equally distributed over the available topics. Each node in the overlay provides four unique services. Using a high number of unique services during evaluation stresses the network and the search algorithms. A low number of services would yield lower bandwidth consumption during evaluation, since less information needs to be propagated. Thus, we chose a high number of unique services to ensure that the results are significant. We set nodes to be interested in the same topics as they offer services in. This choice is based on assumption 4 above; that nodes are more likely to request services within the categories that they are offering.

Bloom Filter: When using the ASAP and Search+ algorithms, the Bloom filter size affects both, the query success rate and the bandwidth consumption. A larger filter increases the success rate, while requiring more bandwidth. We have chosen to have four unique services per node, but in a real life scenario we expect that a few nodes will have significantly more services than others. We also expect that each node may use more than one keyword to describe a service. If we assume

that each node on average will use four keywords to describe a service, and that the node with the most services has 25 services, the Bloom filter should be able to hold 100 keywords for each peer. The filter size of 1000 bit selected for the experiment achieves this goal and results in a probability of a false positive of 0.00819 which we deem to be acceptable. See Appendix B for further details.

Service Discovery: Due to the previously mentioned interest clustering, nodes are assumed to search for services relevant to their interests. Thus, in our evaluation the nodes are configured to search for services that fall within the same categories as they are interested in with a probability of 0.9. In other words, 90% of the queries from each node will be for services that have one of the same topics as the node's own interest. The remaining 10% are chosen randomly.

Network Topology: We expect a tactical network to follow the principle of preferential attachment, where connecting nodes will have a higher probability of connecting to nodes that already have many connections.

In a military network, it is also likely that the nodes will form hierarchical clusters, due to the command structure [17]. Based on these two arguments, we argue that a real life topology form a scale free network with a power law distribution, as described by Barabasi and Alberts in [18]. In such networks, most of the network nodes can be reached in few hops, but complete message dissemination may require a high TTL, as we discuss below. The topology used in the experiments is a 100 node Barabasi-Albert with an average degree of 3.94.

Time-To-Live (TTL): The TTL affects different mechanisms in each algorithm. In Gnutella, the TTL determines how far the flooded query will travel before being discarded. In ASAP, the TTL specifies how far the advertisements are sent. In Search+, the TTL is the number of hops the initial subscription request is forwarded. Due to these differences, the TTL must be chosen independently for each algorithm.

In [19], Portmann et al show that in a power law distributed Gnutella network a TTL of 5 reaches more than 95% of the network. The topology used in our experiments also exhibits power law properties, and we have therefore chosen to use TTL = 5 when using the flooding search algorithm (standard Gnutella search). The initial TTL value for ASAP is chosen based on the results in [15], where they achieve optimal results with a TTL of 6. Search+'s TTL value specifies how far the subscription requests are sent, but after a while the paths that advertisements follow in the overlay may be much longer than this value. We therefore choose the low TTL value of 3.

To examine how the TTL affects the accuracy and bandwidth use, we repeat our experiments with a decreased TTL value for each algorithm.

3.2.2.7 Evaluation

To make it easier to examine the results, the execution is separated into three phases; initialization, stabilization and query. Each phase lasts for a known time interval.
During the initialization phase, the nodes are given 30 seconds to start up, configure themselves and read the topology from a file. After waiting the specified time interval, the peers connect to each other according to the read topology. The overlay is then given 180 seconds in the stabilization phase. After stabilizing, the nodes enter the query phase, and start sending search queries at regular intervals with an added random waiting period. This phase lasts until the experiment ends.

The peers perform a search every 20 seconds, with up to 5 seconds random variation. As a result, the delay between queries varies between 20 and 24 seconds. When entering the query phase, all peers send their first query at the same time. This causes traffic bursts during the first parts of the experiments. After a while, the variation in delay between the queries leads to less traffic bursts, as the queries are distributed more evenly over time between the peers. This gradual change in traffic pattern allows us to see how the algorithms respond during high load, how quickly they recover and how well they function when queries are more evenly distributed.

With an average delay of 22 seconds, each node sends 2.72 queries per minute. For a 100 node overlay, this corresponds to 4.53 queries per second.

Algorithm	Success rate	Response time
Search+, TTL=2	0.929	3.74
Search+, TTL=3	0.974	3.10
ASAP, TTL=5	0.864	2.65
ASAP, TTL=6	0.886	2.57
Flooding, TTL=4	0.823	279.51
Flooding, TTL=5	0.964	230.05

Table 3.1 Success rates and average response times in milliseconds.

Table 3.1 shows the average response times for each algorithm. As is expected, when flooding is used, query times are longest. This is caused by Gnutella nodes having to wait for both the query and the response to be routed through the overlay before a match is found.

In Search+ and ASAP, a confirmation message can be sent directly. Flooding with TTL = 4 has longer response times than with TTL = 5. This is caused by long queues building up in the central nodes during the initial burst of traffic. With TTL = 5, more messages are routed around the central nodes than with TTL = 4, thus leading to shorter response times.

ASAP with TTL = 5 and TTL = 6 have the shortest response times, with 2.65 ms and 2.57 ms. Search+ is a bit slower at 3.10 ms and 3.74 ms, due to the extra calculations required in each node. Still, the result is well below results achieved using flooding.

Table 3.1 also shows the ratio of successful queries. A success rate of 1 means that of all the query responses received, at least one matches the query. As expected, flooding reaches more or less the whole overlay with TTL = 5, and has a high success rate. Search+ with TTL = 3 has the highest success rate with 0.974. ASAP on the other hand, does not quite reach up to the other two, and ends up with a success rate below 90%. Our results with ASAP differ from the results found in the simulation conducted by the authors of [15], where ASAP seems to consistently reach values around 0.95. This can in part be explained by two differences in our setup:

First, in [15] advertisements are distributed with 90% of nodes already connected. This means that the initial advertisement sent when joining the overlay can reach a very high percentage of the nodes, as observed in [19]. This could mean that the success rate of ASAP is largely determined by the initial distribution of advertisements and that the effect of the request message that is supposed to drive advertisements toward interested nodes is negligible. In our experiments we use TTL = 1 for the request message, as recommended in [15], but increasing this value may yield better results. We did not do this in our experiments, mainly because of the additional bandwidth requirements. As we will see later, ASAP already requires the same bandwidth as Search+. The authors of ASAP mention that they performed their experiments after an initial "warm up period", but they do not say how long this period is. We repeated our experiment for 10 hours, but ASAP still did not achieve more than 90% search accuracy. To further increase the success rate, ASAP would probably have to improve its advertisement distribution; either by distributing the advertisements more effectively when joining the network, or by requesting advertisements from more than the immediate neighbors. Either way, ASAP relies on nodes continuing to send queries, as this is what triggers the distribution process.

Second, when simulated in [15], ASAP yields the best results in an actual topology from the eDonkey file sharing network. This topology has 1.28 copies of each document (or service in this context) on average. This would increase the perceived search accuracy, as multiple copies of a document increase the probability of finding one of them. The experiment described in this paper has only one copy of each service.

Algorithm	Advertisement distribution	Average search bandwidth
Search+, TTL 2	23708.67	6385.75
Search+, TTL 3	38763.20	6701.19
ASAP, TTL 5	32406.17	15124.16
ASAP, TTL 6	37375.05	16217.28
Flooding, TTL 4	—	1232089.2
Flooding, TTL 5	—	939114.53

Table 3.2 Average bandwidth consumed (in Kilobytes) measured at the network layer.

Table 3.2 shows the average bandwidth used by each algorithm during our experiment. The results are separated in advertisement distribution (three minutes) and actual search (30 minutes, 4.53 queries per second).

As expected, Gnutella with a flooding algorithm requires most bandwidth, with 1.2GB for TTL = 5. This gives a good query success rate, as we saw earlier, but the amount of traffic will quickly saturate slow links. Both ASAP and Search+ require considerably less bandwidth.

Algorithm	Distribution bandwidth	Search bandwidth	Success rate
Search+, TTL 3	387.63	67.01	0.974
ASAP, TTL 6	373.75	151.24	0.886

Table 3.3 Comparison between ASAP and Search+. Bandwidth is average per node (in Kilobytes).

In Table 3.3, ASAP and Search+ are compared using the TTL values that give the best search accuracy. The bandwidth is shown as average per node in kilobytes. We can see that although Search+ uses slightly more bandwidth during advertisement distribution, the bandwidth used for queries with Search+ is less than half of the bandwidth used by ASAP. This is caused by ASAP continuing to request new advertisements when queries fail. Search+ will only send new advertisements when there have been changes in the overlay, i.e. when new content is published. When comparing the results in Table 3.1 and Table 3.2, we can see that even if we increase the TTL in ASAP, neither consumed bandwidth nor success rate increases significantly. This is probably caused by ASAP relying on distributing advertisements during join, and that even with an increased TTL, the advertisements are not distributed to enough nodes. The bandwidth spent on advertisement distribution directly affects the search accuracy. We therefore expect ASAP to use more bandwidth if the distribution mechanism is improved to increase the success ratio. Search+ has a more aggressive advertisement distribution scheme, which will aggregate advertisements in the overlay and transfer them to new nodes when they start subscribing to topics from their neighbors. Thus, nodes only receive advertisements they have requested. As a consequence, Search+ consumes less bandwidth, while still achieving a success rate of 97.4%. The bandwidth requirements of Search+ can be further reduced by more than 40% with TTL = 2, if we accept a success rate of 92%, which is still higher than we achieved with ASAP. Search+ does however require more processing power at each node.

3.2.2.8 Discussion

Search+ is more bandwidth efficient than the existing solutions that we have evaluated. In our experiments, Search+ with TTL = 3 gave the highest accuracy and the lowest bandwidth consumption of the alternatives. The average bandwidth consumed by Search+ is 387.63 Kilobytes per node during advertisement distribution and 67.01 Kilobytes during search. This corresponds to 17.22 Kbit/s and 0.3 Kbit/s, respectively. The advertisements were exchanged during a period of three minutes. This period could be increased to support lower bandwidths. This shows that it is theoretically possible to use Search+ for service discovery in a disadvantaged grid.

Also, since Search+ runs on top of unstructured rather than a structured P2P protocols we get a solution that is robust and resilient towards attacks and partial failure. The main drawback is that it is, as its other P2P counterparts, based on keyword hashing, which limits the expressiveness of

the search queries. Thus, coupling the solution with a registry service would therefore be required to allow more advanced queries to be executed when needed.

For further details regarding the Search+ design and the experiments we performed, see [20]. More experiments are needed using actual tactical communications hardware before any definite conclusion regarding the suitability of this solution can be drawn. The Search+ implementation is currently being refined and updated with such experiments in mind.

3.2.3 Mercury

To successfully create service discovery for bandwidth-constrained environments, we envision several combined optimizations. For this purpose, we propose a new service discovery solution, Mercury. Mercury describes the service descriptors efficiently as Bloom filters, performs service dissemination by piggybacking service information on OLSR routing messages and utilizes caching of service advertisements.

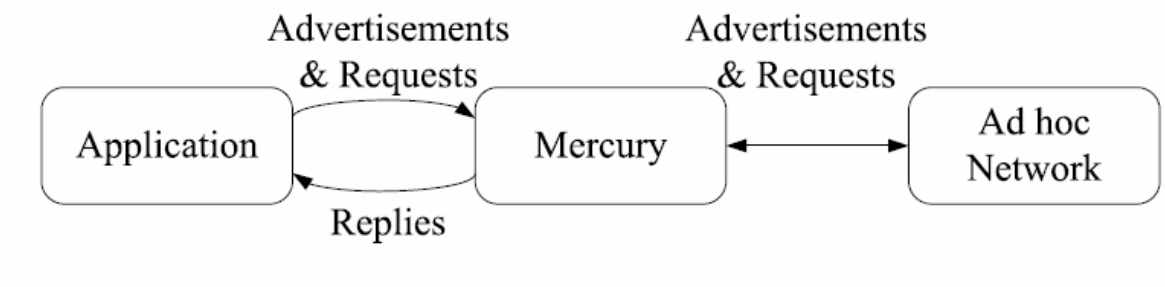

Figure 3.3 Mercury connects users and applications to services in the Ad hoc network using service advertisements and service requests.

Mercury handles requests and advertisements from two entities: (1) Local applications on the node and (2) foreign nodes through the ad hoc network (see Figure 3.3). Each node uses a set of repositories to store the information (see Figure 3.4): Advertised services contain the different services offered by the node itself. The services persist in this list until an upper layer application withdraws the service. Advertisements are sent both when a service is first registered and upon an external request. All the service descriptors in this list are included in advertisements encoded as one single Bloom Filter. In Foreign services cache, all the services offered by other nodes are stored. Each entry in the list consists of the Bloom Filter advertised by the foreign node and its current IP address. The last repository contains the Requested services which stores all the services requested—awaiting an incoming advertisement.

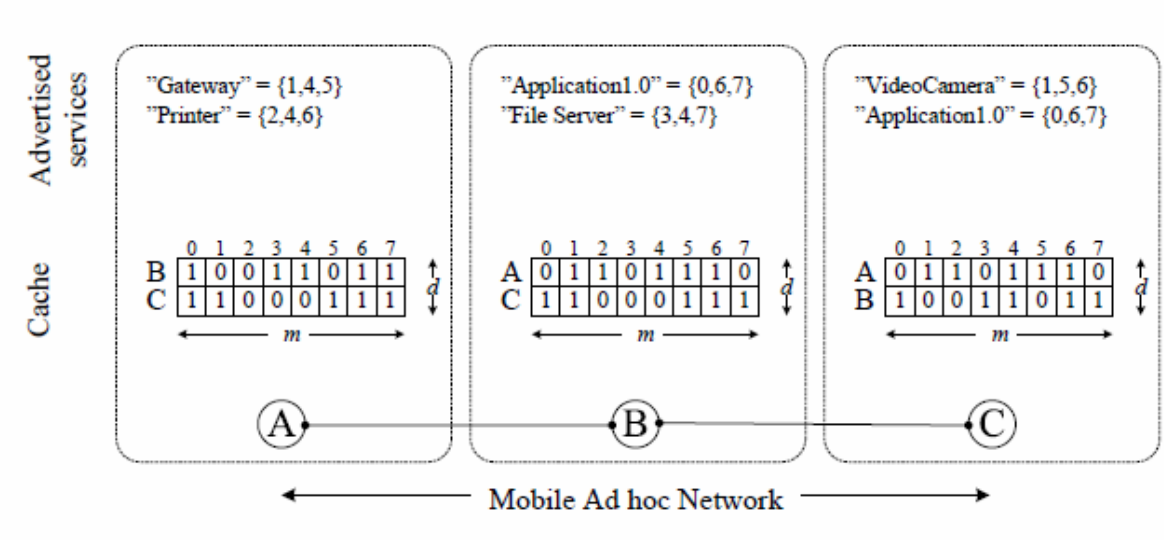

Figure 3.4 A Mobile Ad hoc Network consisting of three nodes. Each node use three hash functions to create the Bloom filter and employs two repositories: One repository stores the local services advertised, and one repository – implemented as an attenuated Bloom filter of depth d – serves as a cache storing advertisements received from foreign nodes.

All incoming advertisements are immediately stored in the cache. Upon a request from an upper layer application, the cache is first requested. If an entry is found, the application is immediately notified. Otherwise, a service request is sent.

3.2.3.1 Protocol format

OLSR communicates using a unified packet format for all data [6]. Using this format the OLSR standard provides extensibility of the protocol without breaking backwards compatibility. This feature gives a unique possibility to disseminate different kinds of information through intermediate nodes even if the nodes do not support the specific extension. We take advantage of the extensibility feature of the OLSR format, and introduce a new message, namely the Mercury service discovery message (MSD). MSD messages are sent as the data-portion of the general message format with the message type set to MSD MESSAGE. The MSD message with the OLSR header has the format specified in Figure 3.5. The Mercury part consists of four fields including a Spare field for future use. The Type field indicates whether the message is a service request or a service reply. The Service Filter field contains the filter describing the services to be requested or advertised encoded as a Bloom filter (described subsequently). The Filter Length gives the size of the filter.

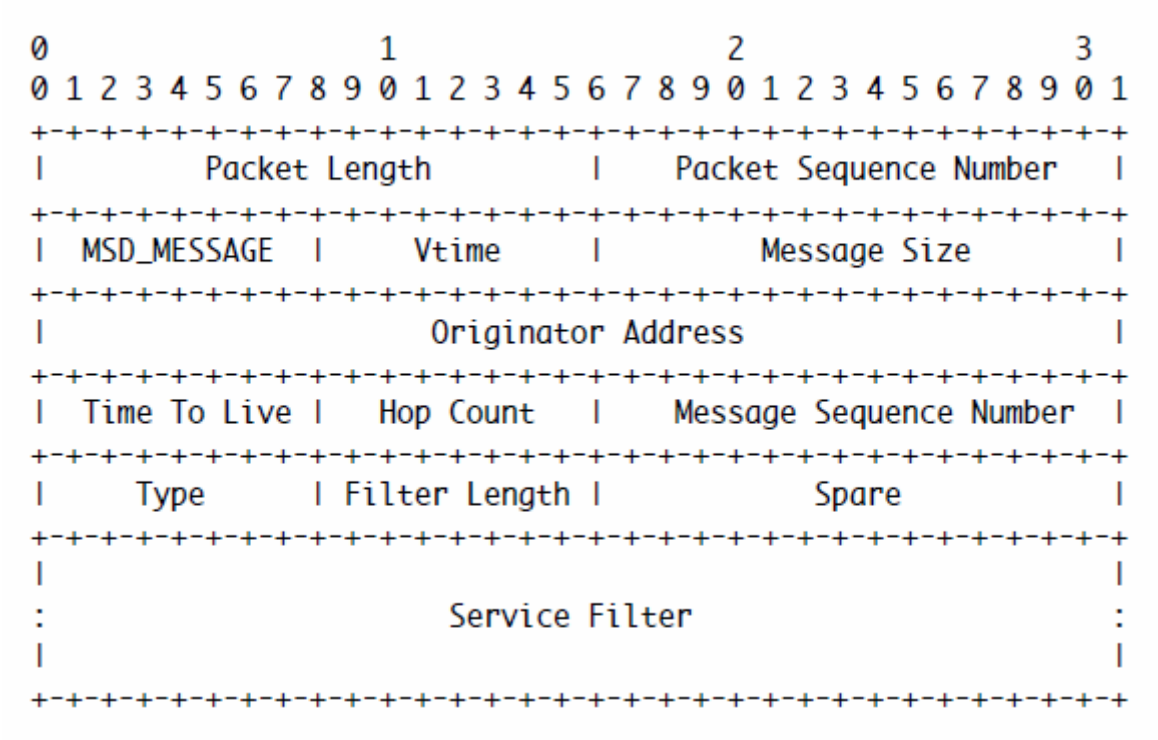

Figure 3.5 Mercury service discovery message format.

3.2.3.2 Distributing service descriptors

Many service discovery protocols use XML to describe the service information, such as in [9]. However, XML requires considerable bandwidth, which is sparse in ad hoc networks. An alternative is to map a predefined set of keywords, or service descriptors, to integers to save bandwidth as proposed in [10]. This solution indeed saves bandwidth. However, it is not very flexible nor is it scalable, as it requires maintenance on every node in the network when new service categories are added. The proposed solution in this paper is therefore to distribute a summary of the available services as a vector described as a Bloom filter [8]. A Bloom filter is a data structure that allows data representation in a simple and space-efficient manner. The filter is created by hashing service descriptors to a size defined bit array. The size limitation may cause the filter to indicate that a service descriptor is in the filter even though it is not—referred to as a false positive. The implementation of the Bloom filter is hence a tradeoff between the size of the filter and the probability of a false positive request to the filter.

In Mercury the filter is created using the message digest function MD5 [11]. MD5 is a cryptographic hash function that hashes arbitrary length strings to 128 bits. A set of k hash functions can then be constructed from k groups of r bits each out of the 128 bit hash. An example usage of Bloom filter based service discovery is shown in Figure 3.4. Each node advertises two services and employs three hash functions to describe the services. After performing service requests, the descriptors are stored in the local cache of the other nodes. The cache consists of one Bloom filter for each of the cached nodes (i.e. attenuated Bloom filter).

3.2.3.3 Caching

Caching is employed to save network bandwidth. Caching may however, lead to false positive replies to the overlying application if the advertised service exists in cache even if the node with the advertised service is — due to network clustering — not available anymore. The cache cleanup timeout is therefore a tradeoff between fast service queries and the false positive rate. To reduce the amount of false replies to the application, we propose a path-aware approach that consults the local routing table for the availability of the nodes in the cache. If a service exists in the cache even if the node is not available, Mercury removes the cache entry and performs a new service discovery in order to find relevant nodes offering a similar service.

3.2.3.4 Implementation and use

The Mercury SD proposal is implemented as an extension to the UniK OLSR implementation (olsrd) [12]. Olsrd supports the loading of dynamically loaded libraries for auxiliary functions using a generic plug-in interface [13]. Here, the Mercury plug-in is briefly described and example usage is given. The code is available at "*http://olsr-mercury.sourceforge.net/*" for further reference. In order to allow communication between the plug-in and user applications, a simple Inter-process communication (IPC) function is enabled via TCP/IP. Using IPC, services are requested, advertised, and withdrawn using a set of simple commands. By using Mercury and by adding only a few code lines, any distributed application can be extended to facilitate service discovery—regardless of programming language.

3.2.3.5 Discussion

Mercury utilizes the default forwarding algorithm in the OLSR routing protocol to disseminate service discovery control messages. This algorithm allows the messages to be forwarded using MPR flooding. When a node sends a control message, the opportunity to piggyback on other messages such as HELLO and TC may be performed to reduce the number of packet transmissions. Mercury takes advantage of caching to reduce unnecessary service requests, and utilizes an optimized way to describe services using Bloom filters. This is a highly optimized protocol, requiring few network resources to disseminate information about available services. Using Bloom filters is very space efficient, but drawbacks are limited expressiveness and the lack of wildcard searches. There is no Web services metadata, since you can search for whole keywords only. For further details about the Mercury protocol, please refer to [21].

Cross-layer approaches demand proprietary solutions and modifications of existing standards, as well as being of limited use if military IPSec solutions are present. These drawbacks have limited cross-layer service discovery from becoming widespread so far, and might limit its usability in a NEC setting. However, cross-layer solutions have low overhead and should thus be investigated for use in military tactical networks where bandwidth is the limiting factor. Application level solutions may also be feasible at this level, since it is possible to reduce the overhead of Web Services significantly by employing XML compression [2]. If one is capable of utilizing an application level solution then that could be preferable, since architecture violations can have a detrimental impact on system longevity, as has been argued for the case of cross-layer design in

[3]. Based on these observations we have created SAM, which is discussed in the following section.

3.2.4 SAM

As we have seen earlier, we lack a suitable Web services standard for use in tactical mobile networks. Ideally, we want to be able to discover Web services in such networks, i.e., employ a mechanism that has richer service descriptions than the ones we discussed above. We have implemented the necessary functionality in an experimental solution which performs distribution of Service Advertisements in MANETs (SAM).

We chose a set of techniques that are well suited for overcoming the service discovery challenges in small, highly mobile networks [36]:

- Decentralized operation to overcome availability issues,
- periodic service advertisements to ensure an up to date view of available services, and
- caching to reduce resource use (no need to query the network).

Further, we introduce piggybacking of NFFI position information to reduce the number of data packets, and compression to reduce the size of the data packets. The Position Data Type (PDT) from NFFI is incorporated in the service discovery mechanism. By distributing the PDT at regular intervals together with service information, the units can know where other friendly units are, and also the services they provide. We implement all these techniques in SAM.

3.2.4.1 Observations and system design

The idea of combining service discovery with blue force tracking is based on the following observations:

- You need to know where your mobile units (e.g. squad members) are, thus there is a need to exchange position data.
- You need to know which services are available where and when, thus there is a need to exchange service data.
- Previously, cross-layer solutions have been suggested for service discovery (e.g. Mercury), but cross-layer solutions violate layered design and are problematic if you want to use IPSec [28]. Therefore, an application level service discovery solution is preferable.
- In some cases it is beneficial not only to discover a service, but also the position of the service. For example, this could be the case if the output of the service is linked to the position of the unit (sensor area coverage, etc).
- Supporting service discovery in the tactical battlefield is a major challenge.
- Since there is a need to send NFFI information at the application layer anyway, we might as well piggyback service information in the notification messages. This gives the benefit of getting the unit position and all active services at the unit in each update.

The information about the currently active services at each unit is described by a WSDL, which is the standardized way to describe Web services. The WSDL contains enough information about a

service not only to invoke it, but also to implement a client to the service. In an operation, this latter function is not important, since each unit will already have the software it needs to fulfill its role. Thus, it is the so-called *endpoint* of the WSDL, i.e. the invocation address of the service that is of interest in an operation. The software needs this endpoint in order to address and invoke the service properly. Since Web services are based on WSDLs, we need to address a few technical issues:

- WSDLs are large. They contain a lot of static information, and some potentially dynamic information. We separate the two, since the endpoint is our main concern.

- The service discovery mechanism should exchange something more compact than the WSDL, since tactical networks have data-rate constraints.

    - Can we distribute a compressed entire WSDL document?

        - The entire WSDL is needed if you want to create new client software for previously unknown services. However, the entire WSDL is not needed at the tactical level, because new clients will not need to be implemented on-the-fly. Units will come with pre-loaded software. The problem here is identifying and invoking known services.

    - Can we use a hash of the WSDL document?

        - A hash will uniquely identify an already known WSDL, and since a WSDL defines a service, it can be used to uniquely identify a service type.

The proposed solution uses a hash over a WSDL with the endpoint removed. Since the endpoint can change, it cannot be a part of the hash. The remainder of the WSDL, on the other hand, is static and identifies the interface of the service. The service discovery mechanism disseminates the hash together with the current endpoint to uniquely identify a service and its invocation address.

3.2.4.2 Implementation

Our experimental service distribution mechanism is tailored to the specific requirements for service discovery in MANETs. It can function disconnected from the Internet since it hosts all the necessary data and metadata in a local repository. It is resilient to partial failure since it is a fully decentralized solution. Thus, it will still function if the network becomes partitioned – the clients in each partition will have a fully operational service discovery mechanism at hand. The mechanism is based on periodic dissemination of service advertisements, and the advertisements are cached locally in each recipient. Whenever a new update is received the cache is refreshed, and any outdated services are removed. There is a limited time each service can exist in the cache before it is deleted, thus ensuring that a query in the local cache will give an up-to-date picture of the available services. An advertisement contains a list of services being provided by the node that sent the advertisement. This list contains entries uniquely identifying the service, and any metadata associated with the service. If the node chooses to report its position (i.e. if it is fitted

with a GPS), then the positional data (e.g. latitude and longitude) is sent along with the list of services.

Basically, an advertisement contains the following XML-encoded data:

- A PDT position from NFFI (optional)
- A list of services, containing one or more of the following entries:
    - Unique service hash ID (required), endpoint URL (required), metadata (optional)

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns2:ServiceDiscovery xmlns:ns2="urn:no:ffi:servicead" xmlns="urn:nato:fft:protocols:nffi13">
  <ns2:position>
    <trackSource>
      <sourceSystem>
        <country>NOR</ country>
        <system>FFI</ system>
      </ sourceSystem>
      <transponderId>BASE1</ transponderId>
    </ trackSource>
    <dateTime>20091116094916</ dateTime>
    <coordinates>
      <latitude>53.3618627551</ latitude>
      <longitude>6.26719155956</ longitude>
    </ coordinates>
  </ ns2:position>
  <ns2:service hash="E67AE486E32BB2FB2F551BD14EC57D56D258AD3C"
address="http:/ / 192.168.3.3:8080/ SamNffiWebService/ SIP3_Service_ReqRes"/ >
  <ns2:service hash="7E0119A31D7244A51C939A46A685271BC82944F2"
address="http:/ / 192.168.3.3:8080/ PingWS/ PingService"/ >
</ ns2:ServiceDiscovery>
```

Figure 3.6 An example advertisement using the mandatory fields and the optional position field.

The *endpoint URL* is the invocation address of the service being identified by the Unique service ID at that particular node. In other words, this is the endpoint address as it can be found in the service definition of the WSDL. This URL is the only dynamic aspect of a Web service description, as the rest of the WSDL can remain static from deployment to deployment. Based on this observation, we found that by removing the endpoint URL from the WSDL, we would get a WSDL which could uniquely describe all services of the type defined therein. A WSDL is a large XML document. To reduce the bandwidth requirements of the advertisement distribution mechanism, we chose to use a SHA-1 hash[3] over the WSDL without endpoint to uniquely identify a service. Metadata associated with the service is optional. It can be used to disseminate additional information which is needed for semantic service discovery. The metadata field is thus

[3] It should be noted that the SHA-1 hash is taken over the entire WSDL (minus service endpoint), and this hash is distributed directly in the service advertisements. This is in contrast to the Search+ and Mercury solutions, where hashes are used to create a Bloom filter. Bloom filters may have false positives, whereas the SAM mechanism will have no false positives from the way the hash is employed. However, a Bloom filter is a more compact way of representing a lot of keywords, and will thus scale to a larger number of services that need to be expressed. Conversely, a Bloom filter is less expressive than hashing the way SAM does, thus making Bloom filter based discovery mechanisms less suitable for Web services discovery.

a flexible measure provided by the solution, in that it can be used by regular Web services for discovery (i.e. just ignore the field) or by semantic Web services for added value and reasoning capabilities. Figure 3.6 shows an example with the mandatory fields of the advertisement being used, as well as the optional position. This is how SAM was used at Combined Endeavor 2009. SAM uses compression to reduce the message size. Currently ZLIB compression is being used. A major issue in dynamic networks is service *liveness*, i.e. if a service has disappeared from the network due to some unforeseen circumstance (e.g. network partitioning, node failure, etc.) the service should no longer be available from the SDS either. Using a registry requires the service to actively de-register by removing itself from the registry. This is one of the main reasons why current Web services solutions are inadequate in MANETs. SAM addresses the liveness issue through caching in the following way:

- Two caches are maintained: A local cache (i.e. a cache of the services published, that is, being provided by this particular unit), and a remote cache (i.e. a cache of services that have been discovered through advertisements sent by other units).
- Updates are sent at regular intervals (this is configurable; during development an interval of 30 seconds is being used). The update is sent in the form of a service advertisement as described above. The list of services in the advertisement is the set of services contained in the local cache.
- Upon receiving an update over the network, the remote cache is updated with the services from the advertisement. The services in the remote cache time out if they are not refreshed through new advertisements. The timeout is configurable, but in the prototype the value is set to two times the advertisement interval, so that the service is deleted from the cache the second time the service misses its advertisement slot. This achieves the desired liveness, in that liveness is defined by the lack of advertisements, rather than requiring the service provider to actively de-register its services. This solution is preferable in a dynamic environment.

Communication in a disadvantageous environment such as a tactical MANET requires an efficient service discovery solution. The proposed solution is resource efficient in that querying for services is always done locally: A lookup in the local cache is all that is needed to get the current state of the network, and no network packets need to be sent to fulfill the request. Also, the advertisements are made as small as possible before transmission by using compression.

3.2.4.3 Evaluation

SAM was designed to be an alternative to WS-Discovery which can be used in tactical networks. Compared to WS-Discovery (we used the draft version of WS-Discovery in our evaluation, since a standard-compliant implementation was not available at the time) SAM is more bandwidth efficient:

WS-Discovery sends “hello” messages during startup. For four web services, this generates 16 packets, i.e. four hello messages per service (the payloads are 1008, 983, 1074, and 1017 bytes respectively). Searching for services involves sending “probe” messages. For each search, four probes are sent (4x 597 bytes payload). WS-Discovery enabled nodes with services then reply by

sending “probe match” messages. In our case one node has four services, resulting in two probe match message (2x: fragmented packets, payload total 2506 bytes). This is not resource efficient in a disadvantaged grid where bandwidth and buffer space limitations come into play.

SAM, on the other hand, sends periodic broadcasts every 30 seconds (this is configurable):

- Four web services *and* position information
  - Uncompressed: 1122 bytes payload
  - Compressed: 728 bytes payload
- Just the four web services
  - Uncompressed: 681 bytes payload
  - Compressed: 516 bytes payload

Thus, we see that SAM is more resource efficient than WS-Discovery.

3.2.4.4 Discussion

SAM combines positioning and blue force tracking updates based on NFFI position descriptions with advertisements of available Web services described by their WSDLs. Our experimental solution is tailored for use at the lowest tactical level, the disadvantaged grids, where disruptions are frequent and bandwidth is scarce. It can be used in environments where the standardized Web services discovery mechanisms are unsuitable, provided that the network supports IP multicast. Hashing, caching and compression techniques are employed to keep bandwidth usage low, while at the same time being able to reap the benefits of XML technology. SAM was the discovery mechanism of choice for use in the Norwegian MANET at Combined Endeavor 2009.

3.3 Conclusion

The table below summarizes our experimental service discovery mechanisms and compares them to the WS-Discovery standard.

	WS-Discovery	**Search+**	**Mercury**	**SAM**
Category	Application layer	Application layer	Cross-layer	Application layer
Service descriptions	Port types and service names	String (Bloom filter)	String (Bloom filter)	WSDL and optional position and metadata
Standardized	Yes	No, experimental	No, experimental	No, experimental
Operation	Fully decentralized or multicast suppression using central discovery proxy.	Current version needs one known node to initiate overlay network. Fully decentralized operation afterwards.	Fully decentralized, but requires OLSR to be the routing protocol.	Fully decentralized using IP multicast.
Suitable for Web services discovery	Yes	Not directly, can be made to work through proprietary implementations	Not directly, can be made to work through proprietary implementations	Yes
Suitable for disadvantaged grids	No	Theoretically	Yes	Yes
Applicable in the Norwegian network at Combined Endeavor?	**Yes**. This Web services discovery mechanism is tailored for LAN usage, and is well suited for use in the deployed HQ. Provides integration with the Norwegian registry by wrapping it in a discovery proxy.	**No**, since it is not tailored for Web services discovery. The lack of rich service descriptions is a drawback. Currently, the implementation relies on a known starting point during startup of the overlay. While this point may be discovered using a second discovery mechanism, it makes Search+ unsuited for use in disadvantaged grids at the present time. The search mechanism is promising, but the P2P startup and overlay building needs further research.	**No**, since it is not tailored for Web services discovery. The solution lacks rich service descriptions.	**Yes**. The mechanism is tailored for disadvantaged grids, where it provides optimized Web services discovery in a fully decentralized manner.

4 Achieving pervasive service discovery

A NEC consists of several different interconnected domains such as strategic networks, tactical deployed infrastructure and tactical mobile networks. There has been much research activity to provide service discovery for each of these network domains. However, as different networks vary in size, equipment, applications and objectives, a variety of service discovery protocols exist to solve specific purposes. Some of the solutions focus mainly on scalability in order to support hundreds or even thousands of nodes. Some solutions seek to minimize latency in the discovery process, while others are focused on reducing the control message overhead to support bandwidth-constrained environments. Further, the solutions are usually aimed at one specific network technology e.g. fixed infrastructure networks or MANETs.

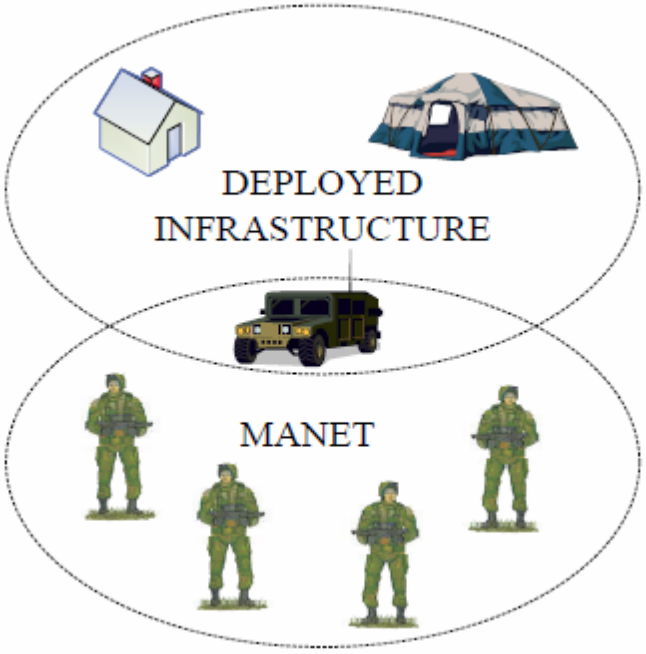

Figure 4.1 Different networking technologies are used at different operational levels. Support for service discovery across network boundaries is needed.

Any service discovery mechanism must take the capabilities and limitations of the target network into account — be it fixed or mobile. Due to the large variation in network capabilities on different operational levels, one single service discovery mechanism cannot be chosen that covers all levels. Thus, there is a need for a toolkit consisting of different service discovery mechanisms so that each network can use the mechanism that is most suited for that particular network. This means that the different service discovery mechanisms must be able to interact with each other. For instance, a protocol in the tactical deployed infrastructure must be able to interact with the protocol chosen in the MANET at the soldier level (see Figure 4.1).

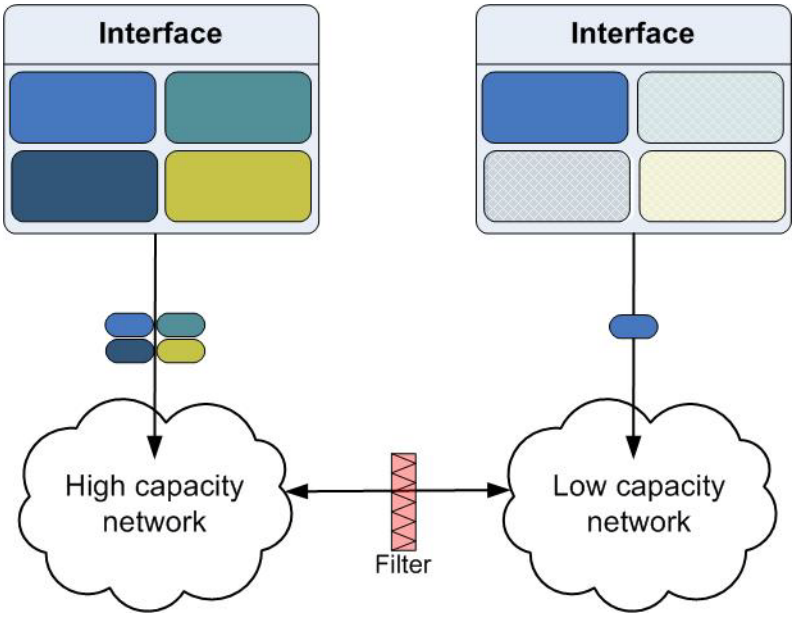

Figure 4.2 Adaptive service discovery protocol.

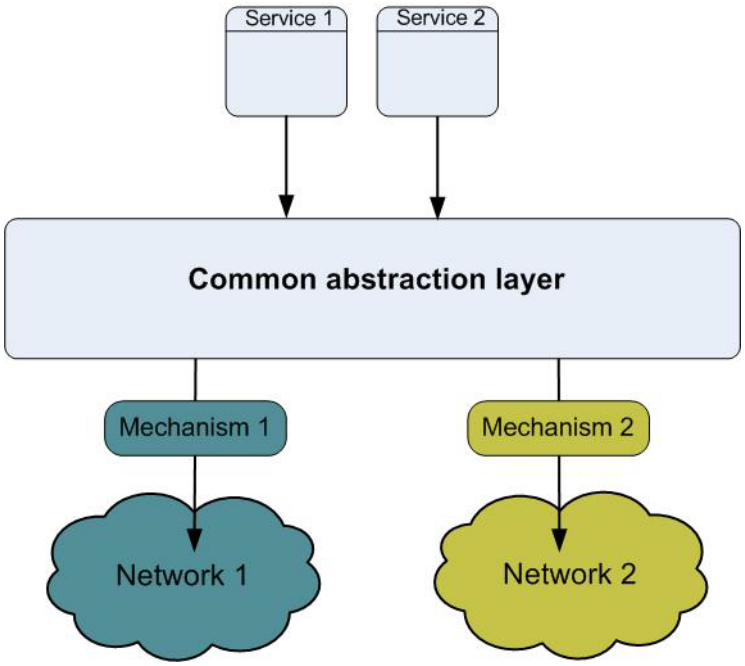

Figure 4.3 Layered service discovery.

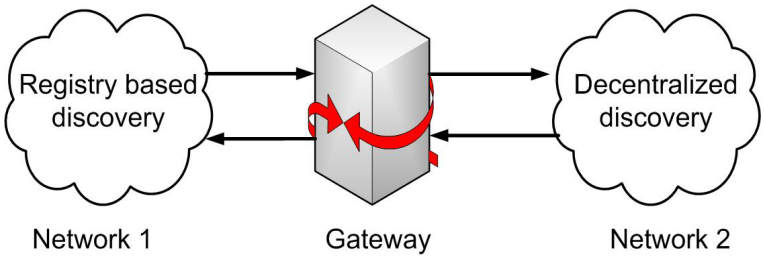

Figure 4.4 Service discovery gateway.

Several methods exist to support service discovery across network boundaries. We classify the different methods in three categories:

1. Adaptive service discovery (see Figure 4.2), meaning that one single service discovery protocol is used across all network domains. The chosen protocol must be able to adapt its behavior according to the limitations and capabilities of each underlying network. All applications in the network must be able to interact with this protocol.
2. Layered service discovery (see Figure 4.3), meaning that each network domain can use a dedicated protocol, but an overlaying protocol controls and connects the different service discovery protocols.
3. Service discovery gateways (see Figure 4.4): Using this method, each network domain can choose the most suitable protocol, and interoperability is ensured by using service discovery gateways between the domains.

Irrespective of the chosen architecture, interoperability is ensured by the creation and interpretation of service descriptions in clients, servers and in gateways. The structure of the different service descriptions determines whether interoperability is fully, or only partially possible.

In the last years, several proposals have appeared to address the challenge of service discovery across different network domains. Some protocols are adaptive and can be used in different domains simultaneously. Bethea et al. investigate the use of ontology-based reasoning for the purpose of developing a general service discovery capability in multi-provider tactical networks [30].

Others involve a layered structure which allows several legacy protocols to coexist by adding a service discovery layer above the others, as presented in [31]. Other initiatives involve service discovery gateways to support transparent interoperability between different protocols, like the scheme of Allard et al. [32]. That scheme allows Jini clients to use UPnP services and UPnP clients to use Jini services, without modification to service or client implementations. A similar study is done by Kang et al. which presents an architecture to provide simple interoperability among various service discovery protocols using a dynamic service proxy [33]. We argue that a gateway based approach is best suited for our purpose as it lowers both the architectural complexity and development cost. Bromberg et al. [31] support our view and conclude that the gateway approach is more efficient than a layered architecture. The use of gateways also makes it possible to use legacy client applications and services unmodified in the network. Our design is therefore based on a gateway design.

4.1.1.1 Service descriptions

A service is a mechanism to enable access to a set of one or more capabilities, where the access is provided by using a prescribed interface and is exercised consistent with constraints and policies as specified by the service description [29].

The Web Services Description Language (WSDL) defines an XML grammar for describing network services as collections of communication endpoints capable of exchanging messages.

WSDL service definitions provide documentation for distributed systems and serve as a recipe for automating the details involved in applications' communication. Thus, WSDLs are a crucial part of Web services, since they define the interface through which you access services, as well as information about where the service can be found in the form of an URL.

A WSDL document uses the following elements in the definition of network services:

- *Types*, which is a container for data type definitions using some type system, such as XML Schema Definition.
- *Message*, which is an abstract, typed definition of the data being communicated.
- *Operation*, which is an abstract description of an action supported by the service.
- *PortType*, i.e. an abstract set of operations supported by one or more endpoints.
- *Binding*, containing a concrete protocol and data format specification for a particular port type.
- *Port*, which is a single endpoint defined as a combination of a binding and a network address.
- *Service*, which is a collection of related endpoints.

```
<?xml version="1.0"?>
<definitions name="PositionUpdate"
             targetNamespace="http://example.com/PositionUpdate.wsdl"
             xmlns:tns="http://example.com/PositionUpdate.wsdl"
             xmlns:xsdl="http://example.com/PositionUpdate.xsd"
             xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
             xmlns="http://schemas.xmlsoap.org/wsdl/">
  <types>
    ...
  </types>

  <message name="GetLastPositionInput">
    <part name="body" element="xsdl:PositionRequest"/>
  </message>
  <message name="GetLastPositionOutput">
    <part name="body" element="xsdl:Position"/>
  </message>

  <portType name="PositionUpdatePortType">
    <operation name="GetLastPosition">
      <input message="tns:GetLastPositionInput"/>
      <output message="tns:GetLastPositionOutput"/>
    </operation>
  </portType>

  <binding name="PositionUpdateSoapBinding" type="tns:PositionUpdatePortType">
    <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="GetLastPosition">
      <soap:operation soapAction="http://example.com/GetLastPosition"/>
      <input>
        <soap:body use="literal"/>
      </input>
      <output>
        <soap:body use="literal"/>
      </output>
    </operation>
  </binding>

  <service name="PositionUpdateService">
    <port name="PositionUpdatePort" binding="tns:PositionUpdateSoapBinding">
      <soap:address location="http://example.com/PositionUpdate"/>
    </port>
  </service>

</definitions>
```

Figure 4.5 An example WSDL file.

Figure 4.5 presents a WSDL file that we have created as an example. It describes a simple service called PositionUpdateService. The types section has been omitted for brevity, but it would contain the data type definitions being used, i.e. a way to express position coordinates, for example longitude and latitude, and perhaps also altitude. The message sections contain information about the data being communicated, in this case two message types are defined: We have a message called GetLastPositionInput and a message called GetLastPositionOutput. The former is used to issue a request to the service, telling it to respond with a position, i.e. with a message of the latter type. This use of the defined messages is described in the portType section, where an operation called GetLastPosition is defined in terms of GetLastPositionInput as the input message type and GetLastPositionOutput as the output message type. In the binding section, we see that the service is bound to SOAP over HTTP, which is by far the most common choice for Web services transport. Finally, a port called PositionUpdatePort is defined using the binding to SOAP, and thus providing the necessary information to use the PositionUpdateService. In

practice, all position services provided in a network would have a WSDL like this, with one important exception - the address location would change for each deployed instance of the service. For this particular example, the address is http://example.com/PositionUpdate/, which points to one deployed position service. In a dynamic environment such as a military ad-hoc network, you can have services that come and go. This means that while all the static metadata are valid the entire time, the address location of the service may change. This means that in such networks, we should be able to discover the current state of the network and find the current addresses of all available services. In other words, we need the ability to map a specific service name to one or more address locations.

4.1.1.2 Static vs dynamic service information

A WSDL is used in the creation of a service and for implementing an interface to the service in client software, but once services are deployed and ready to be invoked only the part of the WSDL that has a potential dynamic dimension to it is necessary. This knowledge allows us to address the challenge of Web services discovery in the tactical domain by utilizing an efficient, proposed solution for service discovery in such networks. Considering that you will use existing software and existing services in the tactical domain, we can assume that exchanging complete WSDL information is not necessary, as long as one can discover the current service addresses. In other words, based on the service name we need to be able to discover the current address location. That means that a simple yet dynamic service discovery mechanism is needed, which can provide the necessary address information based on service name searches.

Thus, we propose to use a service discovery gateway between different operational networks, so that you can get the full Web services information in networks that can handle it, but in disadvantaged grids, where resources are scarce, you can use an optimized solution for service discovery which allows you to discover and invoke known service types.

Mercury is an experimental cross-layer solution for service discovery in MANETs, offering simple string based representation of services. Thus, we use it as an example of a specialized service discovery mechanism that can become interoperable with an existing Web services discovery mechanism through the use of an interoperability gateway.

4.1.1.3 The interoperability gateway

The prototype that we discuss in detail here integrates WS-Discovery with Mercury. This was our original proof-of-concept implementation that was evaluated in the lab. Later the prototype was extended: At Combined Endeavor we used it to interconnect WS-Discovery and SAM, integrating the Norwegian MANET with the Norwegian deployed network. Also, we used it to interconnect SAM with Service Oriented Peers (SOP) (see [37] and [38] for further details) between the Norwegian MANET and the NC3A MANET.

By using an interoperability gateway, service discovery is feasible across network boundaries, connecting mobile soldier systems and deployed tactical systems. Hence, our solution combines the advantages of a directory-based architecture and a distributed MANET architecture. This

gives the advantage of rich service descriptions in the directory-based domain, and the advantage of efficiency and redundancy in the mobile directory-less domain. The transparent gateway between WS-Discovery and Mercury, allows Mercury clients in the soldier network to use WS-Discovery services, and WS-Discovery clients to use Mercury services. No modification of the existing services or clients is needed, as the gateway maintains protocol transparency between the two network domains.

4.1.1.4 Design

The idea is that the gateway resides on a node (i.e. a computer) that is connected to two different networks with two different service discovery mechanisms. It is based on periodic querying for services in each connected network. The gateway is designed as a simple thread-based discovery message router and translator, finding services in one domain, translating the service descriptions from one type to another, and then republishing the service in another domain using its native mechanism. Caching is used to determine if a service is new and should be published, or if a service that was available before has disappeared since the last time the discovery mechanism was invoked. We address these concepts below, where we discuss the details of the implementation.

4.1.1.5 Implementation

Our interoperability gateway prototype is implemented using Java. In our evaluation of the gateway, it was set up on a Linux machine with two network interfaces, one Ethernet interface (eth0) which was connected to the WS-Discovery enabled domain, and one wireless adapter (wlan0) which was connected to the MANET running Mercury, see Figure 4.6.

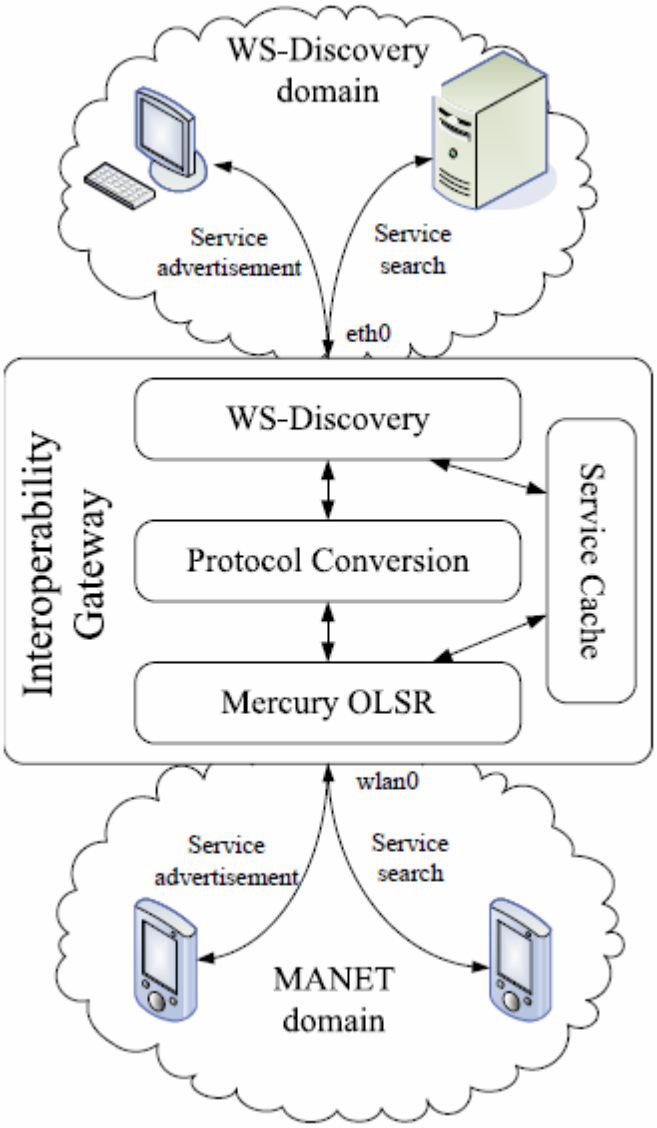

Figure 4.6 *The interoperability gateway connects two different service discovery domains. It allows WS-Discovery clients to use Mercury services, and Mercury clients to use WS-Discovery services. An updated view of the available services in the network is maintained in the cache.*

This setup required one minor modification to the WS-Discovery library, as it was set up to multicast arbitrarily, meaning that WS-Discovery messages would be multicast on both network interfaces. By changing only a few lines of code in the library, we were able to bind the WS-Discovery multicast socket to interface eth0, thereby making sure that WS-Discovery specific traffic was kept on the LAN only. Mercury, due to its tight coupling with the MANET routing protocol, did not exhibit this behavior, and could be used as it was without any modification.

WS-Discovery allows a generic probe (i.e. service query) to be done, and thus the gateway could obtain a complete list of WS-Discovery services from that domain. This list of services could then be parsed, and the service names could be used to publish the services in the Mercury domain.

This conversion was easy to achieve, since WS-Discovery retains a lot of information about each service.

In the Mercury domain, on the other hand, we were faced with two important challenges:

1. The first challenge was related to querying: Since Mercury is based on Bloom filters, you have to search for specific services — you cannot obtain a list of all available services as with WS-Discovery.
2. The second challenge was related to service translation: Mercury describes its services using a simple tuple, i.e. the service name and the IP address where the service can be invoked. This is less information than is needed by WS-Discovery to describe its services. In fact, this is only the dynamic service information.

We addressed these issues as follows: Since Mercury did not support listing all available services, we had to approach that domain differently from that of WS-Discovery. First, we created a list of all names of all supported services in the Mercury domain. Then, we added this list to the query functionality in our gateway, so that instead of issuing a non-specific query, the gateway would search for services from that list in the Mercury domain. This allowed us to find all services in the Mercury domain that had been pre-defined in the gateway. However, the issue of translating these services to WS-Discovery still remained. We needed a way to obtain not only the dynamic aspect of the services, but also the static service information. This meant that our gateway would also have to function as a metadata repository for the known Mercury services, allowing us to supply WS-Discovery with both the static and the dynamic service information that it needed. Having done this, we were able to successfully demonstrate the discovery of — and conversion between — arbitrary WS-Discovery services and a set of pre-defined Mercury services.

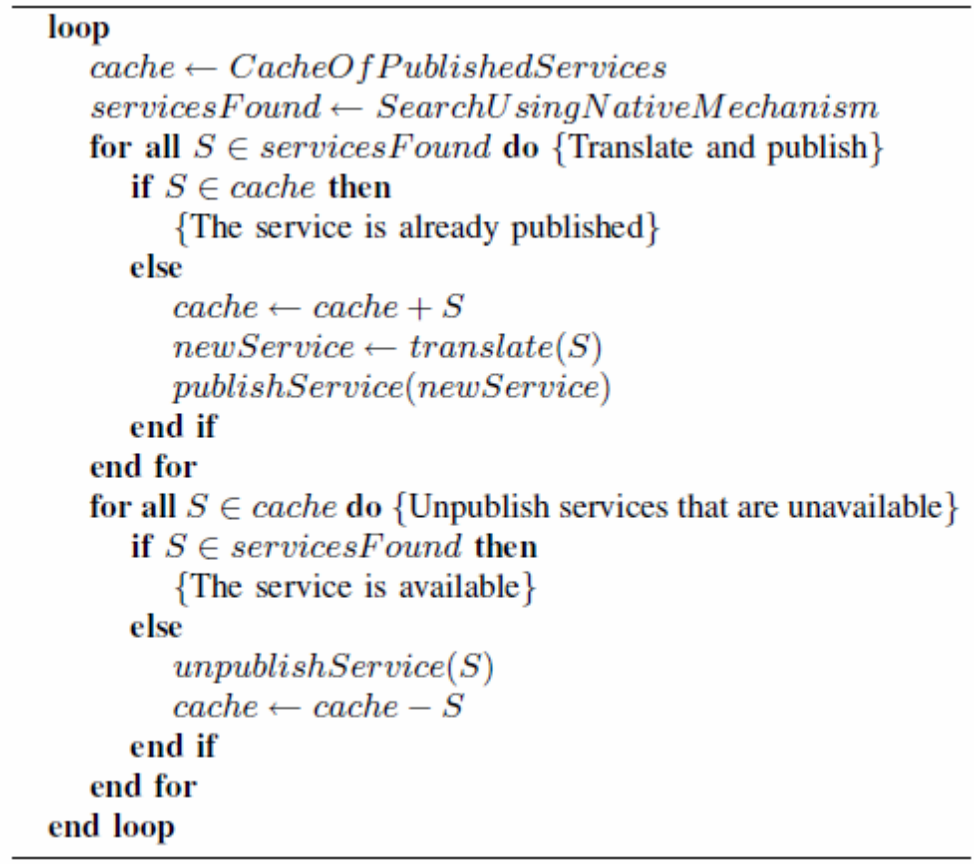

Figure 4.7 Service discovery pseudo code.

The gateway used the algorithm shown in Figure 4.7 for each network interface. Basically, the gateway would periodically (e.g. every 30 seconds, as we used in our evaluation, but this is configurable) query all services in the WS-Discovery domain and the specified list of services from the Mercury domain. The services that were found (if any) would then be looked up in the local service cache. We used the local cache to distinguish between services that had been discovered, converted and published before, and new services that had appeared in each domain. If the service was already present in the cache, then it had been converted and published before, and nothing needed to be done. However, if the service was not in the cache, then it would be added to it. The service would then be translated from one service description to the other, and published on the network. Also, during each query iteration we would compare the local cache containing all previously found services with the list of services found now. If any service had disappeared from its domain since last time (i.e. the service was present in the cache but not in the current set of discovered services), then we would delete the service from the other domain as well by using its native service deletion mechanism. After being removed from the network, the service would also be removed from the local cache. This behavior allowed the gateway to mirror the active services from one domain in the other, and remove any outdated information. Thus, assuming that the service discovery mechanism employed in each domain has an up-to-date view of the services on the network, then this view would be propagated through our interoperability gateway.

4.1.1.6 Discussion

We have considered ways to achieve service discovery interoperability between different operational networks, and have implemented a prototype service discovery gateway. The alternatives considered were adaptive service discovery, layered service discovery and service discovery gateways.

An interoperability gateway is a simple and efficient means of interconnecting two heterogeneous networks, as it provides transparent service discovery description translation from one domain to the other. It is the only one of the three approaches that also has the benefit of allowing legacy clients and servers to function in their respective domains, since it does not require the systems to be adapted to support a new solution. Each system can continue to use the service discovery mechanism that it is designed to use, and that is best suited to its network.

These considerations formed the basis for our decision to pursue a gateway solution. We were able to successfully demonstrate that it is feasible to implement and utilize such an interoperability gateway between two heterogeneous networks using their respective service discovery mechanisms.

At CE, we implemented two such service discovery gateways that we placed in the interoperability points between the networks:

- The gateway between the Norwegian MANET and the Norwegian HQ translated between SAM and the standardized WS-Discovery mechanism. This allowed us to be interoperable with a standard, which through a so-called discovery proxy could provide further integration with the ebXML registry in our HQ.
- The gateway between the Norwegian MANET and the NC3A MANET translated between SAM and the mechanism NC3A used in their MANET, an experimental peer-to-peer based technology called SOP.

5 Registry experiments

During Combined Endeavour 2009 Norway and NC3A conducted experimentation in the area of federating metadata registries based on the Organization for Advancement of Structured Information Standards (OASIS) electronic business eXtensible Markup Language (ebXML) standard, in particular the ebXML Registry Information Model [34] and the ebXML Registry Services and Protocol Specification [35]. These standards are a recommendation by the Information Services Sub-Committee 5 (ISSC/SC5) XML Management Services Working Group (XMLSWG), which was endorsed by the Information Services Sub-Committee (ISSC). Preparation work and experimentation was done via the Core Enterprise Services (CES) test bed.

This section explains the experimentation environment, the federation mechanisms and approaches that were tested and the lessons learned.

5.1 Experimentation environment

The experimentation environment used for pre-testing (i.e. the CES Testbed) was similar to the experimentation environment in Combined Endeavour. The Norwegian metadata registry was connected with the NC3A NATO Metadata Registry & Repository (NMRR) prototype via a switch (see Figure 5.1).

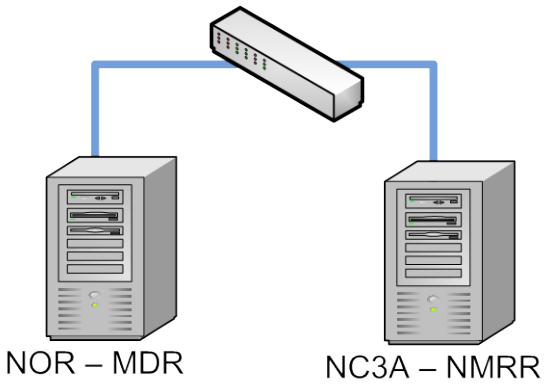

Figure 5.1 Network layout for NC3A-NOR registries federation.

The machines in the test were using the ebXML reference implementation OMAR (also called freebXML), where both the NOR MDR and the NC3A NMRR are built upon OMAR version 3.1.

5.2 Federation mechanisms

The [35] specification describes four different federation mechanisms, i.e.:

- Federated query
- Inter-registry object references
- Local caching of data from another registry
- Object relocation

Due to several challenges, it was decided to focus upon two of these federation mechanisms, i.e., the federated query and the inter-registry object references. Compared to the other mechanisms, the federated query mechanism is foreseen to be used most heavily within an operational metadata registries federation environment.

The federated query mechanism was demonstrated to work in both directions. That is, a query received via the NMRR user interface was federated to the NOR MDR and vice versa. Moreover, the inter-registry object references mechanism was tested. That is, by joining the federation a reference to the NMRR instance was created in the NOR MDR.

5.3 Federation approaches

Before going into the details of the actual experiment, it is relevant to explain the two ways in which a federation can be created and how the membership of the federation is established, i.e. active joining and manual configuration.

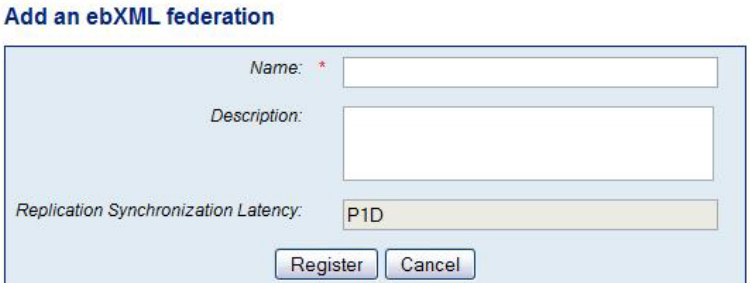

Figure 5.2 Create a Federation object via the NMRR user interface (web-browser).

In both cases, the creation of the federation happens via the user interface (see Figure 5.2), where a user can create a new Federation object by filling out the name, description and replication synchronization latency of the Federation. The latter is related to a period for synchronizing the federation metadata (see [35]).

5.3.1 Active joining (preferred approach)

The preferred approach outlined in the [35] specification is to use the so-called Inter-Registry Object References to refer to remote registries. Instead of creating a local representation of a remote registry, the remote registry should actively join the federation by "*submitting an instance of an Extramural Association that associates the Federation instance as sourceObject, to its Registry instance as targetObject. The home registry for the Association and the Federation objects MUST be the same*" [35]. While this approach is supported by the NMRR user interface, it is not supported by the ebXML reference implementation user interface.

This approach was used to join the NMRR to a Federation in the NOR MDR. It required write access by an NMRR user within the NOR MDR, which was achieved through the following steps:

- Register a user in the NOR MDR (called "WRITE Federate NC3A") via the OMAR web browser interface (NOR MDR)
- Download the p12 certificate.
- Import the p12 certificate in the web-browser
- Import the p12 certificate in the NMRR keystore
- Register the "WRITE Federate NC3A" user in the NMRR
- Classify the "WRITE Federate NC3A" user as RegistryAdministrator in both the NMRR and the NOR.

5.3.2 Manual configuration (alternate approach)

In the manual approach, the joining is done by the administrator of the registry which contains the Federation object (note: in order to perform federated queries in both directions, both registries need to configure the Federation).

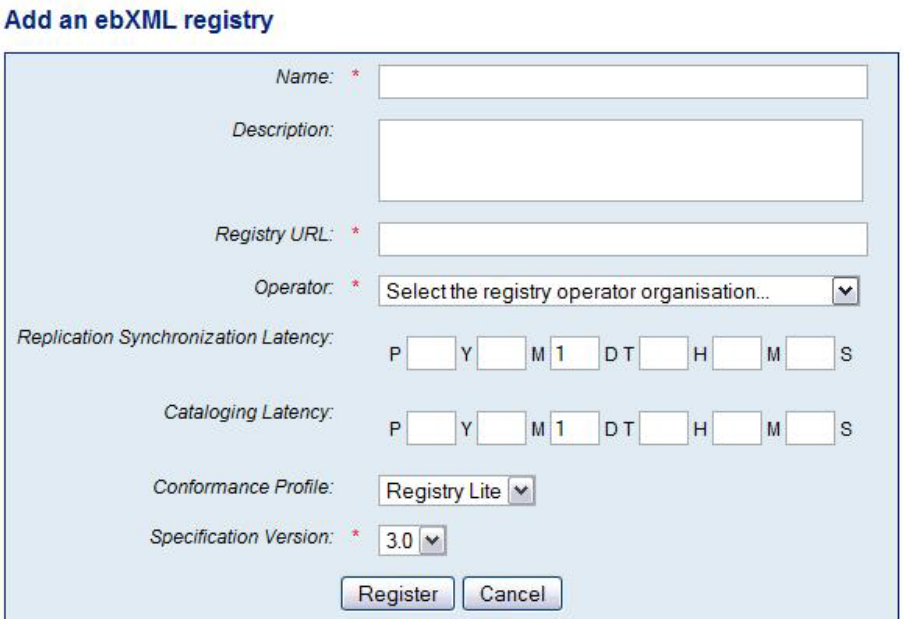

Figure 5.3 Create a Registry object via the NMRR user interface (web-browser).

For each Registry that would like to join the federation, a Registry object has to be created. In the NMRR user interface (web-browser) the Registry object can be created by filling out the name and description of the Registry object, the registry Uniform Resource Locator (URL) (which takes the form http://<hostname>:<port>/omar/registry) and other information as shown in Figure 5.3. Once the Registry object exists, it can be added to the Federation also using the NMRR web-browser interface.

5.4 Lessons learned

This section describes the lessons learned from this experiment.

5.4.1 Federation approach

The experiment has indicated two ways of configuring the federation, i.e. active joining and manual configuration. It is important that the pros and cons of the different federation approaches are well understood. This is considered valuable input for the NATO Metadata Registries Federation Specification, which is being developed under the ACT POW 2009/2010.

5.4.2 Security infrastructure

In order to actively join a federation in the NOR MDR, an NMRR user requires permission to write in the NOR MDR. Currently, each metadata registry maintains its own local keystore with the public keys associated to each of the different user certificates. This way, a user registered in one metadata registry cannot access another registry unless he has registered with the other metadata registry as well.

Instead of duplicating user information among the different metadata registries in a federation, it is recommended to use security services which are separate from the metadata registry web services.

Further research and experimentation is required in this area to determine the details (e.g., which security services are required, what are the interfaces, what are the consequences for cross-security domain interoperability).

5.4.3 Registry profile interoperability

Different metadata registries may apply the ebXML Registry Information Model (ebRIM) in different ways. In other words, they use different registry profiles, which results in a reduced level of interoperability. For example, the artifacts in one registry may be categorized differently than the artifacts in another registry. To ensure that users can still discover and retrieve the metadata that they are looking for, some form of mediation is required among the different registry profiles.

Further research and experimentation is required in this area to determine how mediation between different registry profiles can be achieved in a dynamic and flexible manner.

5.4.4 Unique identification system

Each object in a metadata registry has an identifier that is unique within that registry. As stated in [OASIS ebRS, 2005]: "The *id MUST be a valid Uniform Resource Name (URN) and MUST be unique across all other RegistryObjects in the home registry for the RegistryObject.*" However, the [OASIS ebRS, 2005] does not state that the id should be unique across all registries in a particular federation; neither does it tell whether two objects with the same id residing in different registries should be treated as the same object or as different objects.

Research and experimentation is required to define the best approach for unique identification of registry objects within a federation of multiple registries.

5.4.5 The overall experiment

Lower bandwidth may not be a big issue for Web services, but unreliable connectivity is a problem. This can be mitigated by store-and-forward techniques such as implemented in the DSProxy. However, with the potential for an unstable network, Web services are not suitable for real-time data.

We have seen that service discovery is possible in and across heterogeneous networks. However, by using a transparent gateway to translate between discovery protocols you may lose some service information going from one network to the other. For example, SAM supports both service and position information, but WS-Discovery supports only service information. This meant that our NFFI tracks had to be assembled and built by the gateway, since it was the point receiving the position information. The NFFI tracks could then be exposed as a Web service. The important thing about using gateways for interoperability is that it is sufficient to know the

interface used by another network; you do not need to know the functionality details. This was the case with SOP, where we were able to extract service information, despite being unaware of the NC3A's network topology and how SOP it was deployed in their network.

We noticed some issues when using OMAR, the open source ebXML reference implementation: First, it was not easy to install. You need several old Java libraries to get it to work, since it is incompatible with some of the newer ones. Thus, you need to use exactly the same library versions that are mentioned on the ebXML website. Second, you have to use Sun's own Java. We attempted to install OMAR on a PC using Ubuntu Linux, and the default Java was OpenJDK. That implementation does not implement security, and thus compiling OMAR failed. After uninstalling OpenJDK and installing Sun's own JDK then we were able to compile OMAR. Third, there were issues configuring OMAR properly. OMAR comes with two user interfaces, one Web interface and one Java interface. The Java GUI and the Web GUI support different operation sets. In practice, you need to use both. However, neither of the GUIs set the resource HOME attribute, which is needed in a federation. This attribute tells the registry where the resource belongs. If this attribute is empty, then all responses in a federated query will be treated as if the resources belong to the local registry. If this is not the case, then looking at the XML artifacts will fail, since the identifier will not be resolved to the proper repository's address. To overcome this we had to update the repository database manually, since neither of the provided GUIs supported setting the HOME attribute. This is a hassle, but if you want to use OMAR you have to either live with it or write a new GUI that supports all the necessary functionality. The NC3A had remedied this situation by creating NMRR – their GUI to ebXML.

Our use of registries shows that they can be employed in the deployed HQ, and they can also be used in a federation between HQs. The NC3A has shown that P2P can be employed (i.e., the SOP in their network), and while this technology is mostly suitable in large fairly static networks, it can also be employed to some degree in dynamic networks. In highly dynamic networks decentralized mechanisms should preferably be used, since they address the aspect of service availability and liveness. We addressed these issues by using our experimental SAM mechanism in our MANET. Interoperability between heterogeneous networks and mechanisms can be achieved by

- Using service discovery gateways which translate between discovery protocols.
- Deploying proxies that optimize service invocation across the networks.

The issues we encountered with the ebXML reference implementation clearly show that while standards are important for *interoperability*, the maturity of the available products is equally important for system *usability*.

6 Summary

Standards are important for interoperability between systems from different vendors and nations. For Web services, there exist three standards related to service discovery: The UDDI and ebXML registries, and the decentralized WS-Discovery.

The standardized Web services discovery mechanisms are well suited for use in networks with high bandwidth and fixed infrastructure, whereas experimental solutions must be used in disadvantaged grids. Through our experiments we have shown how interoperability between the experimental and the standardized mechanisms can be achieved using service discovery gateways. By using the gateway approach, each nation can use proprietary solutions suitable for their respective networks, since the gateway can translate to another protocol, thus enabling cross-network service discovery interoperability.

References

[1] R. Faucher et al, *"Guidance on Proxy Servers for the Tactical Edge"*, MITRE technical report MTR 060175, September 2006.

[2] K. Lund et al, *"Using Web Services to Realize Service-Oriented Architecture in Military Communication Networks"*, IEEE Communications Magazine, Special issue on Network-Centric Military Communications, October 2007.

[3] V. Kawadia and P. R. Kumar, *"A Cautionary Perspective on Cross Layer Design"*, IEEE Wireless Commun., vol 12, number 1, February 2005.

[4] I. Chlamtac, M. Conti, and J. J. Liu. *"Mobile ad hoc networking: imperatives and challenges"*, Ad Hoc Networks, 1(1):13–64, July 2003.

[5] C. Perkins, E. Belding-Royer, and S. Das. *"Ad hoc On-Demand Distance Vector (AODV) Routing"*, RFC 3561 (Experimental), July 2003.

[6] T. Clausen and P. Jacquet. *"Optimized Link State Routing Protocol (OLSR)"*, RFC 3626 (Experimental), October 2003.

[7] Kongsberg Defence & Aerospace AS. *"WM600 – Tactical Broadband Wireless Module"*, Datasheet, http://www.kongsberg.com/en/KDS/Products/~/media/KDS/Files/Products/Defence%20Communication/wm600_datasheet_rev_rc_small.ashx

[8] B. H. Bloom. *"Space/time trade-offs in hash coding with allowable errors"*, Communications of the ACM, 13(7):422–426, 1970.

[9] S. Helal, N. Desai, V. Verma, and C. Lee. *"Konark - a service discovery and delivery protocol for ad-hoc networks"*, Proceedings of the Third IEEE Conference on Wireless Communication Networks (WCNC), New Orleans, 2003.

[10] J. L. Jodra, M. Vara, J. M. Cabero, and J. Bagazgoitia. *"Service discovery mechanism over OLSR for mobile ad-hoc networks"*, Advanced Information Networking and Applications, AINA, 2:534–542, 2006.

[11] R. Rivest. *"The MD5 Message-Digest Algorithm"*, RFC 1321 (Informational), April 1992.

[12] olsr.org. *"The OLSR daemon"*, http://www.olsr.org/

[13] A. Tønnesen, A. Hafslund and Ø. Kure. *"The Unik-OLSR Plugin Library"*, In The OLSR Interop and Workshop, 2004.

[14] Clip2. *"The gnutella protocol specification v0.4"*, document revision 1.2. http://www9.limewire.com/developer/gnutella_protocol_0.4.pdf

[15] Peng Gu, Jim Wang, and Hailong Cai. *"ASAP: An advertisement-based search algorithm for unstructured peer-to-peer systems"*, In International Conference on Parallel Processing (ICPP), September 10-14, page 8, Xian, China, 2007.

[16] G. Tyson, A. Mauthe, T. Plagemann, and Y. El-khatib. *"Juno: Reconfigurable Middleware for Heterogeneous Content Networking"*, In 5th International Workshop on Next Generation Networking Middleware (NGNM), September 22-26, Samos Island, Greece, 2008.

[17] Anders Fongen, M. Gjellerud, and Eli Winjum. *"A military mobility model for MANET research"*, In Parallel and Distributed Computing and Networks (PDCN 2009), February 16 – 18, Innsbruck, Austria, 2009.

[18] Albert-Laszlo Barabasi and Reka Albert. *"Emergence of scaling in random networks"*, Science, 289:509, 1999.

[19] M. Portmann, P. Sookavatana, S. Ardon, and A. Seneviratne. *"The cost of peer discovery and searching in the gnutella peer-to-peer file sharing protocol"*, In Proceedings Ninth IEEE International Conference on Networks, 10-12th October, pages 263–268, Bangkok, Thailand, 2001.

[20] Magnus Skjegstad, Frank T. Johnsen. *"Search+: An efficient peer-to-peer service discovery mechanism"*, FFI-rapport 2009/01610.

[21] Joakim Flathagen. *"Service discovery in the soldier networking environment"*, FFI-Rapport 2008/02090.

[22] OASIS. *"Web Services Dynamic Discovery (WS-Discovery)"*. Version 1.1, July 2009. http://docs.oasis-open.org/ws-dd/ns/discovery/2009/01

[23] J. Schlimmer (Editor). *"Web Services Dynamic Discovery (WSDiscovery)"*, Draft, April 2005, http://specs.xmlsoap.org/ws/2005/04/discovery/ws-discovery.pdf

[24] M. Gudgin (editor). "SOAP-over-UDP", 2004, http://specs.xmlsoap.org/ws/2004/09/soap-over-udp/soap-over-udp.pdf

[25] F.T. Johnsen et al. *"Multinett II: SOA and XML security experiments with Cooperative ESM Operations (CESMO)"*, FFI-Rapport 2008/02344.

[26] T. Gagnes. *"Assessing Dynamic Service Discovery in the Network Centric Battlefield"*, Military Communications Conference, IEEE MILCOM 2007, October 2007.

[27] R. Porta. *"Friendly Force Information Sharing – Lessons Learned and way towards NNEC"*, Presentation at the 7th NATO CIS Symposium, Prague, Czech Republic, October 2008.

[28] F. T. Johnsen et al. *"Web services and service discovery"*, FFI-Rapport 2008/01064.

[29] OASIS. *"Reference model for service oriented architecture"*, http://www.oasis-open.org/committees/download.php/16587/wd-soa-rm-cd1ED.pdf, Draft 1.0, February 2006.

[30] W. Bethea et al., *"Automated discovery of information services in heterogeneous distributed networks"*. IEEE MILCOM 2008, November 2008.

[31] Y. Bromberg et al., *"Interoperability of Service Discovery Protocols: Transparent versus Explicit Approaches"*. IST Mobile and Wireless Summit, 2006.

[32] J. Allard et al., *"Jini Meets UPnP: An Architecture for Jini/UPnP Interoperability"*, In Proceedings of SAINT, page 268, 2003.

[33] S.H. Kang et al., *"An Architecture for Interoperability of Service Discovery Protocols Using Dynamic Service Proxies"*, Information Networking, pages 786–795, 2005.

[34] Organization for the Advancement of Structured Information Standards (OASIS), ebXML Registry Information Model, Version 3.0, OASIS Standard, 02 May 2005, http://docs.oasis-open.org/regrep/regrep-rim/v3.0/regrep-rim-3.0-os.pdf (viewed at 13 August 2007).

[35] Organization for the Advancement of Structured Information Standards (OASIS), ebXML Registry Services and Protocols, Version 3.0, OASIS Standard, 02 May 2005, http://docs.oasis-open.org/regrep/regrep-rs/v3.0/regrep-rs-3.0-os.pdf (viewed at 13 August 2007).

[36] Adnan Noor Mian, Roberto Baldoni, and Roberto Beraldi. *"A survey of service discovery protocols in multihop mobile ad hoc networks."* In IEEE Pervasive computing, pages 66-74, January-March 2009.

[37] M. Amoretti et al, *"SP2A: a Service-oriented Framework for P2P-based Grids"*, In proceedings of the 3rd International Workshop on Middleware for Grid Computing (MGC05), Grenoble, France, 2005.

[38] D. Marco-Mompel, *"SERVICE ORIENTED PEER PROTOTYPE FOR MOBILE USERS"*, NC3A Technical Note Draft under project SPW001495, November 2007.

[39] K. Lund, T. Hafsøe, F. T. Johnsen, and E. Skjervold, *"Information Exchange in Heterogeneous Military networks"*, FFI-Rapport 2009/02289.

Appendix A Terminology

Name	Domain	Description
mDNS	NC3A	Multicast DNS
NMRR	NC3A	NATO Metadata Registry and Repository. The term NATO Metadata Registry & Repository is used to denote the overall web-enabled information system, including services, that needs to be developed for storing, controlling and retrieving metadata in a way that satisfies both administrative and operational requirements.
NMRR prototype	NC3A	The NMRR prototype is an ebXML-based implementation used to verify and validate the NMRR specifications.
SDS	NC3A	Service Discovery Service. The Service Discovery Service is a service for discovery of services. It provides visibility and access to information services available in the network-enabled environment.
SDS Prototype	NC3A	The SDS prototype is a JXTA-based implementation used to verify and validate the SDS specifications. To achieve interoperability with other discovery mechanisms it includes bridges for mDNS, UDDI and ebXML.
SOP	NC3A	Service Oriented Peer. A JXTA peer with a GUI that jointly with SDS peer discovers/advertises and invokes web services on the JXTA peer network
NetPeerGroup	NC3A-FFI	JXTA peer group. It will be the virtual network that FFI peers and NC3A peers will connect and communicate trough.
SOP External services	NC3A	Web services outside the JXTA network. For instance WS available in a web server
SOP Remote services	NC3A	JXTA services advertised in another JXTA peer in the JXTA peer group.
SOP Local services	NC3A	Local JXTA services of the peer.
JXTA peer	NC3A	The JXTA peers form a peer-to-peer based overlay network which supports both service discovery and service invocation. Developed by NC3A.
JXTA interface	FFI	Interface to JXTA used by the Service Discovery Gateway to advertise services in the NetPeerGroup.
Registry federation	NC3A-FFI	A registry federation is a group of registries that have voluntarily agreed to form a loosely coupled union. Such a federation may be based on common business interests and specialties that the registries may share. Registry federations appear as a single logical registry to registry clients.
Replication	NC3A-FFI	Operation by which one registry copies services and other data to a target registry so that the target registry can resolve locally queries intended to be resolved by the source registry. Normally this mechanism requires publish-subscribe configuration
Tactical mobile domain/network	NC3A-FFI	Term referring to a mobile or/and dynamic network (MANET) where services are registered/un-registered at unpredictable times.
Tactical deployed domain/network	NC3A-FFI	Term referring to the deployed, wired network where services are meant to have long life.
SD	NC3A-FFI	Service Discovery abbreviation
MANET	NC3A	The tactical network composed by SDS and SOP peers in the JXTA network NetPeerGroup. The NC3A MANET is based on Rajant BreadCrumb nodes.
MANET	FFI	The FFI MANET consists of KDA WM600 radios; several regular nodes and one as a gateway to the NC3A MANET, as it is also equipped with a Rajant BreadCrumb device. This gateway node is also a JXTA peer in the NetPeerGroup joint with NC3A , since the service discovery gateway is running on this node.
Service Discovery Gateway	FFI	The service discovery gateway supports multiple service discovery mechanisms and can translate between them. The gateway is

		necessary because FFI and NC3A use different MANET technologies and different SD mechanisms at the tactical level. Thus, the gateway facilitates interoperability through mutual discovery of services by translating between the SD mechanisms and re-publishing the services using the network native SD mechanism. Developed by FFI.
WS-Discovery	FFI	FFI has implemented the mandatory parts of the Web Services Dynamic Discovery (WS-Discovery) draft specification from 2005: http://schemas.xmlsoap.org/ws/2005/04/discovery/
Mercury	FFI	Mercury is a cross-layer service discovery mechanism for MANETs. It is based on Bloom filters and is implemented as a plugin to the OLSR routing protocol, providing simple string based search and service lookup. Developed by FFI.
SAM	FFI	SAM, short for *Service Advertisements in MANETs*, is an application level discovery mechanism which incorporates positioning and service advertisement dissemination. Developed by FFI.
Service	NC3A-FFI	As defined by OASIS: A service is a mechanism to enable access to a set of one or more capabilities, where the access is provided by using a prescribed interface and is exercised consistent with constraints and policies as specified by the service description.
SOAP	NC3A-FFI	SOAP is the Web services protocol. SOAP is transport protocol agnostic, but is most commonly used over HTTP/TCP. Exceptions exist, for example WS-Discovery uses SOAP over UDP.
UDDI	NC3A-FFI	Universal Description, Discovery and Integration, an OASIS standard for a Web services registry. In this report we do not consider UDDI, but its competing standard: ebXML.
ebXML	FFI	OASIS' other registry standard is electronic business XML. In many ways it is more flexible than UDDI (e.g., better federation mechanism), and since the NMRR is based on ebXML, FFI chose to use this standard for the joint experiments to ease interoperability with the NC3A.
WSDL	NC3A-FFI	The Web Services Description Language is an XML construct which allows service contracts to be made. These contracts define message formats and bindings for SOAP to transport protocols, thus allowing clients and services adhering to the same WSDL to interact in a standardized and interoperable manner.
Search+	FFI	Search+ is a peer-to-peer based service discovery mechanism. It is based on Bloom filters, providing simple string based search and service lookup. The mechanism is subscription based, thus reducing overall communication needs. Developed by FFI.

Appendix B Bloom filters

A Bloom filter is a hash based data structure that provides a membership function with a certain probability of false positives, never false negatives. Bloom filters were first described by Bloom in [8].

More specifically, a Bloom filter comprises an array of bits and a number of independent hash functions. When a data element is inserted into the filter, the hash functions are used to calculate a set of hash values representing the data element. For each hash value, the corresponding bit is set in the array.

To check whether a Bloom filter contains a specific data element, the process is the same, except that the generated hash values are compared to the existing array instead of being stored. If all the bits corresponding to the different hash values are true, then the data element can be determined to have been stored in the Bloom filter with a given probability.

A small Bloom filter example is shown in Figure B.1.

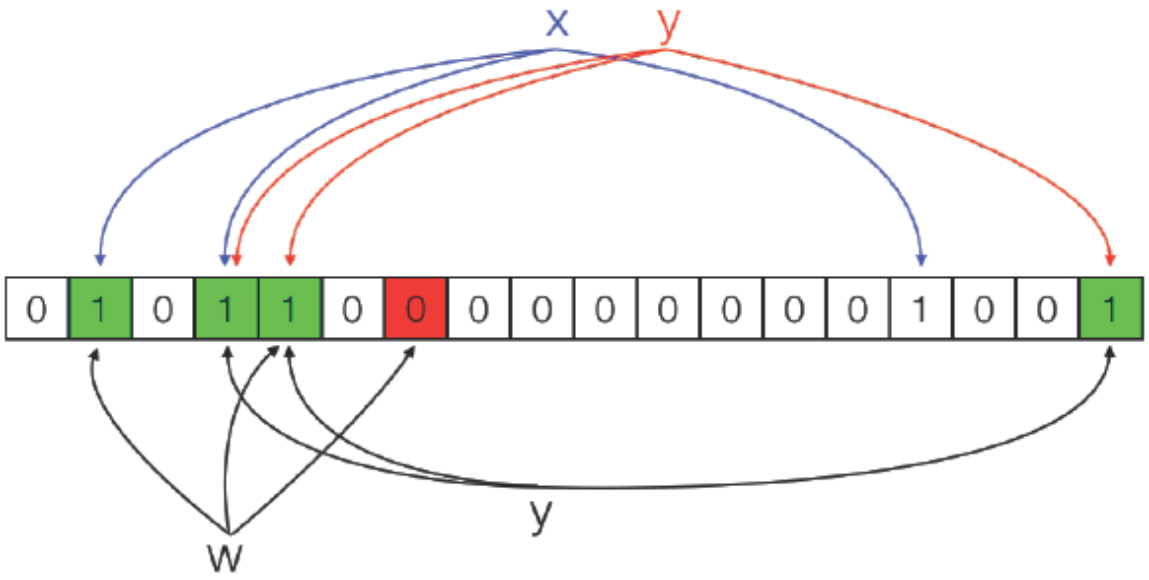

Figure B.1 A Bloom filter with $k = 3$ and $m = 18$. Data elements x and y are inserted into the bit array — one bit is set for each hash function. Elements w and y are checked against the filter — y is present with a certain probability, w is decidedly not since one of the bits is false.

The probability of a false positive in a Bloom filter is the same as the probability of all the bits for a data element already being set. We can calculate the probability p that one specific bit is not set by a specific hash function in a bit array of size m with:

$$p = 1 - \frac{1}{m}$$

Further, we can calculate the probability p of a specific bit not being set by k hash functions after inserting n elements with:

$$p = \left(1 - \frac{1}{m}\right)^{kn}$$

Therefore, the probability p that k hash functions set k specific bits to true after inserting n data elements can be calculated with:

$$p = \left(1 - \left(1 - \frac{1}{m}\right)^{kn}\right)^{k} \approx \left(1 - e^{-k\frac{n}{m}}\right)^{k} \tag{B.1}$$

Equation B.1 gives us the probability that the bits corresponding to the hash values of a new data element are already set to true— or in other words, the probability of false positives.
Ideally, p should be kept as small as possible, but it can also be desirable to keep k or m small, to save computing resources or storage capacity — or in our case, bandwidth. Intuitively, we can see that p will increase when n increases, and decrease when m increases. A high m=n will give a smaller probability of false positives. The optimal number of hash functions k for a number of given data elements n in a bit array of size m can be determined by:

$$k = \frac{m}{n} \ln 2 \tag{B.2}$$

In our experiments we wanted to have a Bloom filter with low probability of false positives with 100 stored elements. We chose n = 100 and m = 1000, with m=n = 10. Using Equation B.2, we find k = 6.93, which we round up to 7. The probability of false positives can then be calculated with Equation B.1, giving p = 0.00819.