Paper ID 900608

SEARCH+: A RESOURCE EFFICIENT PEER-TO-PEER SERVICE DISCOVERY MECHANISM

Magnus Skjegstad
IFI, University of Oslo
Norway

Utz Roedig
InfoLab21, Lancaster University
United Kingdom

Frank T. Johnsen
Norwegian Defence Research Establishment
Norway

ABSTRACT

A recent feasibility study suggests that a Service Oriented Architecture (SOA) will be central within the NATO Network Enabled Capabilities information infrastructure. An important element of a SOA is the service discovery process, but existing solutions are not sufficient in military networks. We suggest to use a robust Peer-to-Peer (P2P) network as a complement. Unfortunately, available P2P search algorithms have high bandwidth requirements which cannot be supported by low-bandwidth links in tactical networks.

This paper describes a new search algorithm named ***Search+*** *which is designed for tactical networks with limited bandwidth resources. The evaluation of Search+ presented in this paper shows that it outperforms existing search algorithms in terms of bandwidth consumption while achieving comparable search success rates.*

INTRODUCTION

A Service Oriented Architecture (SOA) is characterised by its loose coupling, standardised protocols and that it is web-enabled. As a result, a SOA is very flexible and can handle situation and environment changes which is useful in a military context. Appropriate solutions for interaction between service consumers and producers in a military SOA exist (see [1]). However, the service discovery mechanisms that are currently available for SOAs are not ideal for use in tactical environments.

The most common approach to implementing a SOA is by using Web services. Existing Web services discovery solutions such as UDDI registries [2] or the ebXML registries [3] are based on a central registry being available, thus introducing a single point of failure. It is possible to increase robustness by configuring the registries in a so-called *federation*, in which several registries cooperate by replicating data between them. This approach removes the single point of failure but introduces a second severe problem. Registry replication itself requires a lot of bandwidth which makes this solution undesirable at the tactical level where bandwidth is scarce [4]. However, it has to be noted that registry replication is a viable option on a strategic level as bandwidth is not a limiting factor.

In this paper we propose a novel service discovery approach for SOAs which can be of use in certain military networks. We propose to implement service discovery using a Peer-to-Peer (P2P) overlay network at the tactical level, where the existing standardised solutions are not ideal. P2P networks do not have a central entity and therefore no single point of failure. In addition, we propose a novel P2P search algorithm named *Search+* for service discovery implementation. Search+ is designed to conserve scarce network resources in tactical networks. The specific contributions described in this paper are:

- *Search+ Definition:* A detailed specification of the Search+ algorithm is presented.
- *Search+ Evaluation:* An implementation of a P2P based service discovery mechanism is used to compare Search+ with other existing search algorithms.

P2P BASED SERVICE DISCOVERY

The term *peer-to-peer* 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 applications.

As mentioned before, 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 fulfil 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 neighbouring 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 neighbouring nodes. The exact forwarding behaviour of request messages is again dependent on the used P2P algorithms. 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 introduces again single points of failure. In fact, such systems are similar to a federation of central registries as they are used in non-P2P SOAs. Examples of structured P2P networks are *Pastry* [5], *Chord* [6], *CAN* [7] and *Tapestry* [8].

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 neighbouring 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* [9].

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 *ASAP* 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.

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 neighbours together with a *time-to-live* (TTL) value. The neighbours will decrease the TTL value and forward the message to their neighbours, 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 (see evaluation in the next section).

ASAP

Advertisement-based Search Algorithm for Unstructured P2P Systems (ASAP) [10] is an algorithm specifically developed to reduce bandwidth consumption and improve search accuracy in unstructured P2P networks. The algorithm is considered *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 are 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 — there different units will 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 [11] 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[1] 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 neighbours. Over time, a node builds up a local registry that is able to answer a nodes service discovery requests.

The nodes in the overlay synchronise their registries by sending registry updates whenever their own registry (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. The authors of ASAP performed simulations for evaluation. 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.

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:

[1]Two service descriptors might have the same bits set in the Bloom filter. By defining the size of the filter and the number of hash functions, the probability of a false positive can be determined.

- 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.

a) Join: When a new node joins the network, the algorithm described in Algorithm 1 is executed. Its task is to inform the neighbouring 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 sent. Note that Search+ does not send service advertisements at this point.

Algorithm 1 The Search+ join algorithm

```
//Send a subscription request to the new neighbour
I ← getOwnInterests()
N ← new node
sendSubscriptionRequest(N, I, ttl)
```

b) Maintenance: The maintenance process in Search+ has two main functions; to *process* newly arrived subscription requests and to *publish* service advertisements.

In Algorithm 2, 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 neighbouring 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 neighbours. When a node receives a subscription request message with TTL greater than 0 it adds its own interests and its other neighbours' interests to the message before forwarding it.

New advertisements are published as described in Algorithm 3. The algorithm steps through the list of neighbours and checks 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.

Algorithm 2 The Search+ algorithm executed while processing newly arrived subscription requests

```
//Process all subscription requests
for all r ∈ newly received requests do
  N ← getSourceNode(r)
  requestedInterest ← getInterests(r)
  ttl ← getTTL(r)
  storeSubscription(N, requestedInterest)
  if ttl > 0 then
    totalInterest ← ∅
    for all interest ∈ active subscriptions do
      totalInterest ← totalInterest ∪ interest
    end for
    forwardSubscriptionRequest(totalInterest, ttl − 1)
  end if
end for
```

Algorithm 3 The Search+ algorithm used to publish newly received updates

```
//Publish new advertisement to interested neighbours
for all s ∈ active subscriptions do
  N ← getSubscribingNode(s)
  I ← getNodeInterests(s)
  sentAds ← getSentAdvertisements(N)
  updatedAds ← getAdvertisementsMatching(I) ∉ sentAds
  sendUpdatedAvertisements(N, updatedAds)
  storeLastSentAdvertisements(N, sentAds ∪ updatedAds)
end for
```

c) Search: The search process is described in Algorithm 4. 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 fall back 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 behaviour distinguishes Search+ from ASAP which requests new advertisements from neighbours 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.

Algorithm 4 The Search+ search algorithm.

```
//Search for match
I ← getOurInterests()
K ← getSearchTerms()
R ← ∅ {R will contain the results}
for all ad ∈ localAdsCache do
  B ← getBloomFilter(ad)
  if match(K, B) then
    S ← getAdSource(ad)
    sendConfirmationRequest(S, K)
    if isConfirmed(S, K) then
      R ← R ∪ S
    end if
  end if
end for
return R
```

IMPLEMENTATION AND EVALUATION

To evaluate the proposed Search+ protocol we implemented a P2P overlay. We used the Juno [12] 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 [9]. 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 are interested in seeing if Search+ can improve the search hit rate while reducing bandwidth consumption.

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.

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 topics that their own offered services are in as well. This choice is based on the assumption 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 $1000bit$ $(128byte)$ 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.

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 [13]. 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 [14].

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 [15], 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 [10], where they achieve optimal results with a TTL of 6.

Search+'s TTL value specifies how far the subscription requests will be 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.

Evaluation

To make it easier to examine the results, the execution is separated into three phases; *initialisation*, *stabilisation* and *query*. Each phase lasts for a known time interval.

During the initialisation 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 stabilisation phase. After stabilising, the nodes enter the query phase, and start sending search queries at regular intervals. 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 bursty traffic, 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.

TABLE I
SUCCESS RATES AND AVERAGE RESPONSE TIMES IN MILLISECONDS.

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 I 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 6 has the shortest response times, with $2.65ms$ and $2.57ms$. Search+ is a bit slower at $3.10ms$ and $3.74ms$, due to the extra calculations required on each node. Still, the result is well below results achieved using flooding.

In Table I, the ratio of successful queries is shown. A success rate of 1 means that all queries received at least one match. 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 [10], where ASAP seems to consistently reach values around 0.95. This can in part be explained by two differences in our setup:

First, in [10] advertisements are distributed with 90% of nodes already connected. This means that the initial advertisement sent on join can reach a very high percentage of the nodes, as observed in [15]. 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

TABLE II
AVERAGE BANDWIDTH CONSUMED IN KB.

Algorithm	Adv. distr.	Search
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

advertisements toward interested nodes is negligible. In our experiments we use $TTL = 1$ for the request message, as recommended in [10], 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 [10] 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 neighbours. Either way, ASAP relies on nodes continuing to send queries, as this is what triggers the distribution process.

Second, when simulated in [10], ASAP yields the best results in the crawled topology. 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 increases the probability of finding one of them. The experiment described in this paper has only 1 copy of each service.

Bandwidth Usage: Table II 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.

In Table III, ASAP and Search+ are compared using the TTL values that give the best search accuracy. Bandwidth is shown as average per node in kilobytes. We can see that although Search+ uses slightly more bandwidth

TABLE III
COMPARISON BETWEEN ASAP AND SEARCH+. BANDWIDTH IS AVERAGE *per node* IN KILOBYTES.

Algorithm	Distr. bandw.	Search bandw.	Success
Search+, TTL 3	387.63	67.01	0.974
ASAP, TTL 6	373.75	151.24	0.886

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 I and Table II, 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 neighbours. 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.

Findings

As Search+ with $TTL = 3$ gives the highest accuracy and the lowest bandwidth consumption, it is the most suitable candidate for search in low bandwidth networks such as tactical networks. The average bandwidth consumed by Search+ is $387.63KB$ per node during advertisement distribution and $67.01KB$ during search. This corresponds to $17.22kbit/s$ and $0.3kbit/s$, respectively.

In our experiments the advertisements were exchanged in a three minute period. This period could be increased to support lower bandwidths. This shows that it is the-

oretically possible to use Search+ for service discovery in a tactical network.

RELATED WORK

P2P networks have in fact been proposed as means of implementing service discovery in many non-military systems to improve reliability. For example, the projects SP2A [16], WSPeer [17] and PWSD [18] use P2P networks for service discovery. However, these described solutions cannot be directly used in a military context as additional requirements must be fulfilled. In particular, tactical networks have severe bandwidth limitations which need to be taken into account. Existing non-military solutions are designed to operate on the Internet, and are thus not designed to conserve bandwidth. P2P networks have been proposed for military networks [4]. However, this existing work does not provide the necessary implementation details. For example, no specific search algorithm is described that can be used for an implementation of the service discovery mechanism.

CONCLUSION AND FUTURE WORK

Our solution is more bandwidth efficient than the existing solutions that we have evaluated, proving it to be a strong candidate for a service discovery mechanism in tactical networks. Also, since we base our solution on an unstructured rather than a structured P2P protocol 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. We are planning to implement this connection to a Web services registry in the near future. Also, when time and resources permit it, we will evaluate our solution in an actual operational setting.

ACKNOWLEDGEMENTS

We would like to acknowledge Gareth Tyson at InfoLab21 and Trude Hafsøe at FFI for valuable feedback during the experimentation and writing process.

REFERENCES

[1] K. Lund, A. Eggen, D. Hadzic, T. Hafsøe, and F.T. Johnsen. Using web services to realize service oriented architecture in military communication networks. *Communications Magazine, IEEE*, 45(10):47–53, October 2007.

[2] F. Curbera, M. Duftler, R. Khalaf, W. Nagy, N. Mukhi, and S. Weerawarana. Unraveling the Web Services Web, An Introduction to SOAP, WSDL, and UDDI. *Internet Computing, IEEE*, 6(2):86–93, April 2002.

[3] Sally Fuger, Farrukh Najmi, and Nikola Stojanovic, editors. *ebXML Registry Information Model v3.0*. OASIS ebXML Registry Technical Committee, May 2, 2005.

[4] Tommy Gagnes. Assessing dynamic service discovery in the network centric battlefield. In *Military Communications Conference*, pages 601–614, Orlando, Florida, USA, Oct 2007.

[5] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, volume 11, pages 329–350. Heidelberg, 2001.

[6] I. Stoica, R. Morris, D. Karger, M.F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 149–160. ACM New York, NY, USA, 2001.

[7] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker. A scalable content-addressable network. In *Proceedings of the 2001 SIGCOMM conference*, volume 31, pages 161–172. ACM New York, NY, USA, 2001.

[8] B.Y. Zhao, J.D. Kubiatowicz, and A.D. Joseph. Tapestry: An infrastructure for fault-resilient wide-area location and routing. Technical report, Technical Report UCB//CSD-01-1141, UC Berkeley, 2001.

[9] Clip2. The gnutella protocol specification v0.4, document revision 1.2. http://www9.limewire.com/developer/gnutella_protocol_0.4.pdf, visited September 9th, 2008.

[10] 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.

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

[12] 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.

[13] 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.

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

[15] 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.

[16] M. Amoretti, F. Zanichelli, and G. Conte. SP2A: a service-oriented framework for P2P-based grids. In *In proceedings of the 3rd International Workshop on Middleware for Grid Computing (MGC05), Grenoble, France*, 2005.

[17] Andrew Harrison and Ian Taylor. WSPeer - An Interface to Web Service Hosting and Invocation. In *HIPS Joint Workshop on High-Performance Grid Computing and High-Level Parallel Programming Models, IPDPS*, Denver, Colorado, USA, 2005.

[18] Y. Li, F. Zou, Z. Wu, and F. Ma. *PWSD: A scalable Web Service Discovery architecture based on peer-to-peer overlay network*, pages 291–300. Springer Berlin / Heidelberg, 2004.