DISSERVICE: A PEER-TO-PEER DISRUPTION TOLERANT DISSEMINATION SERVICE

Niranjan Suri[1], Giacomo Benincasa[1], Steve Choy[2], Stefano Formaggi[1], Mirko Gilioli[1], Matteo Interlandi[1], Jesse Kovach[2], Silvia Rota[1], Robert Winkler[2]
[1]Florida Institute for Human & Machine Cognition, Pensacola, FL
[2]U.S. Army Research Laboratory, Adelphi, MD

ABSTRACT

Tactical networking environments demand reliable, robust, and efficient approaches to disseminating information that are tolerant to unreliable and bandwidth-constrained networks. This paper describes DisService, an Agile Computing approach to information dissemination that opportunistically discovers and exploits excess communications, storage, and processing capacity in a distributed network to improve the performance of information dissemination. DisService is disruption tolerant and caches data throughout the network by replicating the data. Nodes subscribe to hierarchically organized groups. Information is published in the context of a group, and may also be tagged to differentiate between multiple types of data (e.g., blue-force tracking, sensor data, logistics, or other runtime information). Each node operates in a distributed, peer-to-peer manner while processing and communicating the published information and requested subscriptions from neighboring nodes. Information is disseminated using an efficient combination of push and pull, depending on the number of subscribers, the capacity of the network, the stability of nodes in the network, and the predicted information needs of users. Finally, DisService also supports efficient dissemination of large data by replicating and scattering fragments throughout the network. These features combine to realize an effective approach to information dissemination for tactical networks.

1. INTRODUCTION

Tactical military environments are often highly dynamic and consist of fixed and mobile nodes such as unattended ground sensors (UGS), robots, dismounted soldiers, ground vehicles, and airborne nodes such as UAVs. These nodes are often interconnected with wireless networks that result in unreliable and bandwidth-constrained links. Moreover, power constraints, especially for battery-powered devices, require efficient use of processing, storage, and communications. In spite of the challenges posed by these environments, users demand reliable, robust, and efficient approaches to information dissemination.

This paper presents the design and implementation of DisService – an information dissemination service that is part of the Agile Computing Middleware. DisService opportunistically discovers and exploits excess communications, storage, and processing capacity in a distributed network to improve the performance of information dissemination. It supports the storage and forwarding of data and caches data throughout the network, thereby making it disruption tolerant and improving the availability of data. Information is published in the context of a group, and may also be tagged to differentiate between multiple types of data (e.g., blue-force tracking, sensor data, logistics, or other runtime information). Each node in the network running DisService operates in a distributed, peer-to-peer manner while processing and communicating the published information and requested subscriptions from neighboring nodes. DisService disseminates information using an efficient combination of push and pull mechanisms, depending on the number of subscribers, the capacity of the network, and the stability of nodes in the network. DisService takes into account the user's preferences in order to anticipate information requests as well. In addition it proactively fragments large data objects and replicates the fragments in order to improve sharing and increase availability of large data objects.

This paper is organized as follows. Section 2 presents three motivating scenarios for data dissemination in a tactical environment. Section 3 describes the DisService capabilities. Section 4 presents the DisService architecture and design. Section 5 highlights the unique design features of DisService. Section 6 presents the evaluation of DisService performance. Finally, section 7 presents conclusions and discusses future work.

2. MOTIVATING SCENARIOS

The emerging tactical environment consists of assets with vastly different computing, communications, storage, and survivability capabilities. At one extreme, low-power miniaturized Unattended Ground Sensors (UGS) offer minimal computing, communications, and storage capabilities and can survive months or even years in a hostile environment. At the other extreme, large tactical vehicles are capable of carrying racks of

978-1-4244-5239-2/09/$26.00 ©2009 IEEE

high-performance servers and fast long-range radios but are large, loud, and can survive only as long as their fuel supply. These nodes have network connectivity that is limited and intermittent, caused by lack of radio coverage, resource contention, or the desire to operate in a clandestine manner and maintain radio silence. Given these constraints, providing a high operational time in tactical missions is a significant challenge. While on a mission, soldiers need to access a variety of information including maps, aerial reconnaissance, various sensor data, intelligence reports, and blue and red force tracking. Some of this data may be preloaded onto the nodes and some may become available later. This information has been classified as follows:

Situational awareness data needs to be disseminated as widely as possible throughout the network while minimizing redundant transmissions and latency using an intelligent flooding strategy. SA data includes blue-force tracking data showing the location and intent of friendly forces as well as fused sensor data that has been used to generate enemy location reports.

Directed data is of interest to only a single node or a small number of nodes. This type of data needs to be delivered to its destination as quickly and efficiently as possible. This data may also need to be held for future transmission if the destination is currently unreachable. Directed data can include raw sensor data which needs to be processed and correlated with data from other sensors at a fusion station.

On-demand data is data that is large or otherwise problematic to transmit and is only of interest to some nodes some of the time. To minimize impact on the network, this data should be delivered only to interested nodes on an as-needed basis. Metadata describing each data item needs to be disseminated as situational awareness data. This class of data includes images and video collected by sensors, unmanned vehicles, or analysts.

These three classes of information are treated in the following evaluation scenarios:

2.1 Harvester scenario: In this scenario, a number of different sensor fields connected by low-bandwidth, power-saving radios are deployed within an area of interest. While most of these devices have extremely limited storage and communication capabilities, one is designated to be a gateway and is equipped with greater storage and communication capabilities. As data is collected by each of the sensors within the field, it is directed to the gateway where it is held until a harvester (potentially residing on a ground vehicle or UAV) comes within range to collect the data. The sensor gateway then dumps all of the sensor data (which consists of both raw sensor detections and multimedia images or video) to the harvester. The harvester, equipped with extra storage and long-range radios, is often in contact with a sensor fusion station. When a fusion station is reachable, the harvester sends all of the raw sensor data to the fusion station using the directed data dissemination strategy. The harvester also disseminates the metadata associated with the imagery and video collected by the sensor fields in a manner that covers as many nodes as possible.

2.2 Prediction scenario: In this scenario, a team of dismounted soldiers are carrying nodes preloaded with information relevant to their mission. Once deployed in theater, the system will need to automatically retrieve new information that is relevant to the team and to each member of the team. To this end, any information gathered from ground (or other) sensors that are along the path being traversed are deemed essential. New data may also become available either back at the operations center or from some other unit in theater. In long duration missions it is also possible to have changes in the mission itself and in the orders and consequent adjustments to planned activities. In dynamic environments, the system must disseminate on-demand data in an efficient and timely manner. However, in the instance that an environment is observed with degraded communications, both the source node and the requesting node may be offline or may not have sufficient bandwidth to communicate at the request time. To compensate for this problem, the system uses a learning algorithm to predict the user's requests and proactively moves information ahead of time to the user's node or to a nearby node, depending on the resources available.

2.3 Large objects scenario: Large objects are on-demand data such as images or videos that are only needed some of the time. Generally, they are held by gateway nodes until requested. These nodes publish large objects spreading their associated metadata in the network. The metadata might be received by an analyst at a Tactical Operation Center, who might be interested in the associated large object and decide to retrieve it. While the basic mechanisms to spread the metadata of SA data and of directed data are similar, other strategies are needed to ensure efficient distribution and storage of large objects in the network. Despite DisService's primary purpose not being file sharing, similar techniques that enable DisService to achieve timely retrieval and balancing of the storage resources are being investigated. As the large object is routed back to the

Tactical Operation Center, intermediate nodes cache it to satisfy other future requests. With this strategy, data that is requested more often will become more available over time. An alternate strategy is for the gateway node to pre-replicate the data when the network is idle, which will increase the availability and reduce the latency for data access even for the first request.

3. DISSERVICE CAPABILITIES

DisService provides the capabilities necessary to disseminate information between peer-to-peer nodes interconnected in a network. The following subsections describe the key components: subscription management, pushing data, information anticipation, and large object dissemination.

3.1 Subscription management: DisService applications disseminate data in the context of a group, so they must subscribe to a group before they receive any data. They may dynamically subscribe and unsubscribe to groups as necessary. Each subscription may have an associated priority indicating the application's preference among multiple subscriptions. In situations where the bandwidth is limited, the priority can be used to order the dissemination of data. Subscriptions may also request sequenced and/or reliable delivery of messages. Messages published by a node, in the context of a group, are sequenced. A subscriber may request that this sequence be maintained when delivering messages to the application. This ensures that messages will not be delivered out of order. If reliable delivery is requested, missing messages are requested to be retransmitted. Sequencing and reliability may be combined together. Reliable subscriptions that are not sequenced imply that all the messages will be delivered, but possibly out of order. On the other hand, unreliable but sequenced subscriptions imply that messages may be missing, but all the delivered messages will be in sequenced order.

3.2 Pushing data: An application wishing to disseminate data can use two different approaches: pushing data or publishing data to other nodes. With the first approach, the data is delivered to applications that have subscribed to the corresponding group. On the other hand, publishing data implies that only metadata describing the data is pushed to subscribing nodes and each node that wishes to receive the data has to explicitly request it. Messages that have been pushed are received and delivered asynchronously to applications. Publishing data is normally used for large items, such as multimedia objects, which would be too expensive (bandwidth-wise) to send to an application without ensuring that it really desires to receive the object.

DisService automatically caches the messages received by nodes. Messages may be expired and deleted, depending on the actual resources available on the node and on parameters of the message such as the expiration time, and the history window. The expiration time associated with a message indicates a time interval since the information was published, and represents the amount of time during which the information is relevant. The history window indicates the number of previously published messages for each group that should be cached. Cache replacement strategies play an important role when resources are limited. So far, DisService implements a simple strategy solely based on the expiration time. While more accurate strategies are being investigated, DisService leaves the possibility to the application to create its own domain-specific cache replacement strategy. This can be done by overriding the default DataCacheExpirationController (see section 5.3).

Finally the message priority identifies the relative priority of one message with respect to other messages being pushed. The priority plays an important role when there is not sufficient bandwidth to transmit all the requested data. In this situation DisService chooses to first transmit data with high priority.

3.3 Information anticipation: In an environment with degraded communications, the user node may be offline or may not have sufficient bandwidth to retrieve the information after a user request. Information anticipation, where the system can predict the user requests and pre-stages the data ahead of time, increases information availability. In order to predict what data each node will need in the future, the system needs to have access to metadata about the users and the data currently present in the network. Part of this information is stored in the metadata fields associated with each data, and part of it in a model of the user context. The metadata field is enhanced with additional information that describes the content of each data with some characteristics relevant in a tactical military environment. The user context contains information relevant to the local user (for example role and actual position) and the mission. This information depends on the particular mission in progress, so it can be dynamically changed at any time. Given that the system has access to the user's context, it can predict the set of possible information the user may request. Therefore, the system has a rudimentary form of predictive capability, in that it can automatically select certain types of data to be pushed to a user. However, these constraints alone may not be strong enough to produce acceptable system performance. To improve the performance, the system needs a learning capability in order to adapt itself to a particular user's preferences. At any time, the system accesses the metadata associated with the data currently stored in the local cache and infers the history of past requests made by the user. After that the system uses a modified version of the C4.5 machine learning algorithm [6] to look for patterns in usage history and predicts which information the user is likely to request in the future.

3.4 Large object replication: DisService provides replication capability for large objects. Replication is performed as a proactive procedure, in order to reduce the impact of network partitions on data availability. By replicating data, the overall redundancy of data is increased in the whole network. This makes DisService more resilient to node access problems, since nodes can access data even in the presence of network partitions. To this end, DisSevice replicates large data objects in the network as soon as possible, taking into account the current available resources (i.e. bandwidth and memory capacities). Large object replication is performed by replicating pieces of a large object. In DisService these pieces are referred to as chunks. These are smaller and can be easily duplicated and transmitted over links where disconnections occur frequently. In addition, chunk sharing between nodes is easier than large objects sharing, since nodes can process and transmit chunks as soon as they have been received. In this way large objects will be disseminated easily in the network.
In order to reconstruct the original large objects, DisService provides a chunk recovering mechanism to reassemble each large object. This mechanism uses some functionalities of the Group Manager [1] to do a peer-to-peer search for nodes which hold chunks that make up the desired data object.

4. DISSERVICE ARCHITECTURE AND DESIGN

DisService provides efficient and peer-to-peer dissemination of data without any reliance on centralized components. There are no assumptions made about the presence of stable network connectivity. Instead, DisService dynamically adapts itself to network changes and disseminates information as best as possible. Figure 1 shows the DisService architecture. The DisService components are described in the following subsections.

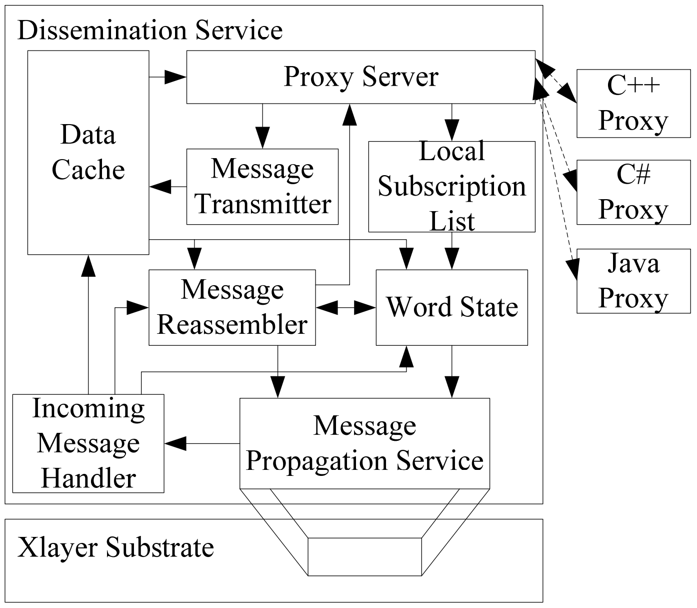

Figure 1: DisService Architecture

4.1 Message Propagation Service: This service provides two capabilities for efficient use of bandwidth: message consolidation and piggybacking. The first capability allows DisService to send multiple individual messages that are automatically consolidated into network packets in order to minimize the number of packets injected into the network. The consolidation capability allows DisService to include a delay tolerance in each message transmission request. This specifies how long the message should be kept in order to consolidate it with other messages. In addition message consolidation allows DisService to send multiple, small messages without having to worry about the number of packets being generated. This is particularly important for some packet rate limited radios.

4.2 Data Cache: The design choice for DisService is to aggressively cache data on every node, limited only by the local storage capacity. Therefore, any data that has been previously pushed or received by a node is held in the Data Cache. This allows the node to readily provide the data both to local applications as well as any peer nodes that need the data. The current implementation of the Data Cache uses the SQLite library [5], a public domain embeddable SQL database library.

4.3 Message Transmitter: The Message Transmitter handles fragmentation of large messages and controls the bandwidth utilization of outgoing traffic. Messages that are larger than the Maximum Transmission Unit (MTU) are automatically fragmented. Each message contains a header that identifies the portion of the data contained in the message. Messages that are transmitted may be rate-limited in order to not overload the network. When multiple messages are awaiting transmission, the message priority is used to determine the transmission order.

4.4 Incoming Message Handler: Messages received by the Message Propagation Service are handled by the Incoming Message Handler, which examines the header to identify the nature of the message. An incoming message may contain payload data or control data. For payload data, the handler checks if the whole data is present in the current message or the data has been fragmented. Fragmented data is handled by the Message Reassembler. Before delivering the data to the application, the sequencing rules for the subscription are checked. If sequencing has been requested, an out of order message is not delivered. After that, the message is delivered to the correct applications.

Control messages are handled differently. Two types of them are possible. A World State message includes information about a neighbor node and is handed off to the local World State component. The second type is a Data Request message. When a data request arrives, the Incoming Message Handler checks both the Message Reassembler as well as the Data Cache. This because partially received messages are stored in the Message Reassembler until they are complete before being stored in the Data Cache. In situations where only partial data is available, the local node will transmit the subset of data that is available. This contributes to satisfy the request.

4.5 Message Reassembler: This component takes incoming data fragments and reassembles them in the correct order. If reliable delivery has been requested by the subscribing application, the Message Reassembler identifies missing fragments, requests them from other nodes, and performs the reassembly procedure when all the missing fragments have been received. If an application has subscribed to the group with a request for sequenced delivery, two possibilities exist. If reliability is requested too, messages received out of order are buffered until the missing messages are received and then delivered in order to the application. If no reliability is requested, old messages that are received later are simply dropped. That is, delivery of a message with sequence number *n* ensures that no message prior to sequence number *n* will ever be delivered. The Message Reassembler periodically checks for missing messages or messages that have missing fragments and sends out a request to the peers for retransmission of the missing fragments.

4.6 World State: The World State maintains the best known information about the state of other nodes in the network, including the messages that they contain. Given the distributed nature of the system, this information might not be accurate and up to date. Each node maintains its own view of the world in the local World State component. As part of the World State, each node maintains information about local neighbors and their subscriptions, as well as all known remote nodes. The information held for a remote node includes the distance to the remote node (in terms of the number of network hops), the path to the remote node, and the lowest link capacity, which limits the overall bandwidth available to that node. A sequence number is attached to each World State, which is incremented at every change to the World State. Periodically each node will broadcast its presence and the sequence number of its World State. This broadcast is received by all the peer nodes, which may request the complete World State if the node is new or if the sequence number has been updated. This reduces unnecessary transmissions of the World State.

4.7 Proxy Server: The Proxy Server allows multiple applications on the same node to utilize DisService. Proxy libraries connect to the Proxy Server via TCP socket connection, thereby allowing different languages to be used for the proxy libraries. The implementation of the core DisService uses C++. To date, proxies have been implemented using C++, C#, and Java. A client application simply utilizes the Proxy Library, which transparently manages the connection to the actual DisService instance. While applications and proxies normally reside on the same node as DisService, this architecture also supports remote proxies. This usage pattern is useful to support resource constrained devices, which may only run the proxy and connect to the actual DisService instance elsewhere. For example, small sensor nodes may run the proxy and connect to DisService running on a sensor gateway node.

5. UNIQUE FEATURES OF DISSERVICE

DisService design and implementation incorporate some unique features, which are highlighted in this section.

5.1 Neighbor Dependent Probabilistic Response Model: An important aspect of DisService is that a request for data may be received by multiple peers, which act independently from each other. In such cases, the receiver may get multiple responses for the same data request, wasting network capacity. In order to reduce this duplicated traffic, the probability of a node responding to a request is computed based on the number of neighbors of the requesting node. Each node maintains the number of its neighbors. When a node transmits a request for data, the request is received by all the neighbors. All of them potentially reply and transmit duplicate copies of the data. To avoid this, the requesting node includes its neighbor number, in the data request. Each node will then transmit with a probability that is the inverse of the number of neighbors. It is also possible that some nodes don't have the data sought. For instance, if just one neighbor has the data, the requesting node may not receive the data sought, due to the nature of the probabilistic response model. To alleviate this situation, when a node sees a repeated request for data, the probability measure is ignored and the requested data is always transmitted.

5.2 Reliable Reception Instead of Reliable Transmission: When reliable data transmission is desired, traditional network protocols such as TCP rely on the sender to ensure that the data being transmitted is received by the recipient. The DisService model differs significantly from the point-to-point model assumed by TCP. The first difference is that communication in DisService is point-to-multipoint: several peer nodes may be recipients of some data transmitted by one node. The second difference is that each recipient may independently request or not request reliable reception of data. The third difference is that the set of nodes that are reachable may change continuously. For instance, if a node is pushing information to another, and the second node moves away and loses network connectivity with the pusher, the TCP model would result in having the pusher node continuously attempting to retransmit data to the receiver. In the DisService model, when the receiver moves away, the pusher node does not care. Eventually, if the receiving node comes in contact again with the original pusher or some other node which has the messages, then the receiver will request and receive the missing messages at that time. Another difference is that the recipient node does not need to go back to the original transmitting node to get the missing information. This is because it may be obtained from any other peer node that has the desired information. This is an effective strategy for reliable delivery, especially when coupled with the design choice to aggressively cache as much data as possible at each node.

5.3 Customizable Controllers: A feature of the DisService architecture is to enable customizability of the default behavior of DisService. To this end, the architecture defines three customizable interfaces, which may be implemented by a system to modify the default behavior of DisService. These interfaces are the ForwardingController interface, the DataCacheReplicationController interface, and the DataCacheExpirationController interface. Default implementations are provided for each of these interfaces. However, applications and systems that desire to modify the default behavior can define their own controllers that override one or more of these defaults.

6. PERFORMANCE EVALUATION

The performance of DisService is evaluated using two different experiments. The first experiment measures the performance in the context of the data harvesting scenario. The second experiment measures the performance in the context of disseminating SA data (blue force tracking information) between a set of peer nodes. In both cases, the baseline performance was obtained using TCP/UDP-based applications. NISTNet [8] was used to control the reliability of network, varying from a completely reliable network to a network with a 20% packet loss probability. While DisService still functions at higher packet loss rates, TCP was not practical with a 30% or higher packet loss rate, given its exponential back-off algorithm.

6.1 Data Harvesting Experiment: This experiment consisted of three nodes – a sensor gateway, a data harvester, and a receiver, as shown in Figure 2. The sensor gateway holds gathered sensor data. Periodically, a harvester node comes into contact with the sensor gateway and retrieves all acquired data. When the harvester subsequently comes into contact with the receiver, the gathered data is uploaded.

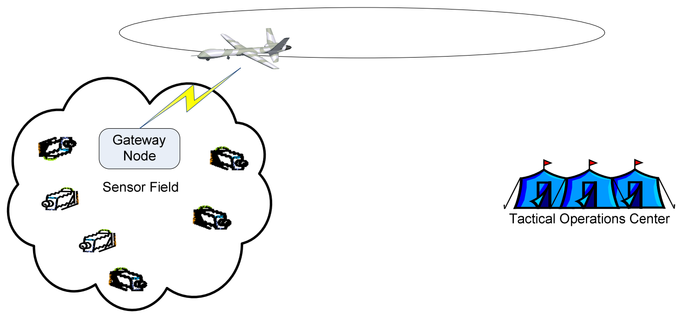

Figure 2: Data Harvesting Experiment Scenario

The results are shown in Tables 1 and 2. Three different link bandwidths were used – 10 Mbps, 500 Kbps, and 250 Kbps. Table 1 shows the overhead for each approach. Table 2 shows the time for the data transfer to be completed. In all cases, the gateway node contained 1284 KB of data that was harvested and delivered to the receiver.

As the results show, DisService does have more overhead than the TCP-based application. Note that this comparison is not entirely fair, since the TCP-based application is simplistic. For example, when the Harvester comes into contact with the Gateway, the Harvester simply gathers all available data. No protocols exist that check and verify whether the data has already been acquired (for example, as part of a previous mission). Hence the TCP results are not necessarily useful for direct comparison, but provide a baseline case nonetheless. Also note that the TCP-based application does not perform well when the network reliability is 80% or lower, which is quite possible with wireless tactical networks.

Table 1: DisService Performance (Overhead) for Sensor Harvesting

	Overhead (Percentage)					
	DS	TCP	DS	TCP	DS	TCP
Reliability	(10 Mbps Band.)		(500 Kbps Band.)		(250 Kbps Band.)	
100%	10.97	6.32	8.63	6.3	9.08	8.67
90%	25.5	21.4	33.35	20.62	51.99	21.55
80%	39.71	42.94	58.2	35.16	106.53	
70%	57.21		80.05		144.39	
60%	83.6		86.57		167.39	
50%	129.43		129.35		161.34	

In the case of the completion time measure (Table 2), the results show that the TCP-based application outperforms DisService only in the case of a fully reliable network, and with 90% reliability with a 10 Mbps connection. In all other cases, DisService outperforms TCP. In fact, when reliability was less than 80%, the TCP test was terminated if it failed to complete within 30 minutes (1800 seconds).

6.2 Data Dissemination Experiment: This experiment consisted of eight nodes exchanging SA data in a peer-to-peer configuration. The scenario is shown in Figure 3. Each node generates a 512 byte message at a rate of 1 Hz that is received by all the other nodes.

The TCP-based application in this case used a centralized redirector that receives messages from each node and reflects it back to every other node. This architecture is based on the Tactical Object Server (TOS) used by the Army Research Laboratory. The DisService application being peer-to-peer does not require any centralized server. Also, to make the comparison fair, the no network disconnection was introduced in this scenario, since the TCP application would not have been able to cope with the disconnection and would result in lost messages.

Table 2: DisService Performance (Completion Time) for Sensor Harvesting

	Completion Time (Seconds)					
	DS	TCP	DS	TCP	DS	TCP
Reliability	(10 Mbps Band.)		(500 Kbps Band.)		(250 Kbps Band.)	
100%	10.65	7.25	29.23	27.87	50.85	47.81
90%	28.44	21.87	46.05	50.03	63.75	73.86
80%	50.78	497.04	51.39	1485.56	81.37	
70%	70.18		77		97.49	
60%	102.87		119.48		119.92	
50%	143.61		170.17		142.5	

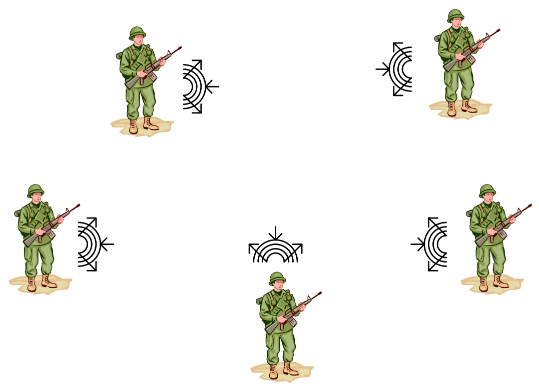
Figure 3: Data Dissemination Experiment Scenario

The results are shown in Table 3 and highlight the advantage of the peer-to-peer nature of DisService. The TCP-based application has to transmit each update eight times over the network, once from the node generating the message to the centralized redirector, and seven times from the redirector to the each of the other nodes. The DisService-based application performs better since it uses broadcast to reach as many node as possible during each transmission, and only retransmit when nodes do not receive the message directly.

Table 3: DisService Performance (Bandwidth) for Data Dissemination

	(Total Bandwidth)		(Bandwidth Rate)	
Reliability	DS (KB)	TCP (KB)	DS (KB/s)	TCP (KB/s)
100%	6101.68	23073.39	9.71	38.44
90%	10847.51	25956.77	17.16	43.20
80%	13234.57	27945.89	20.73	36.12

7. CONCLUSIONS AND FUTURE WORK

This paper described the motivation, design, and implementation of DisService – a peer-to-peer data dissemination component. DisService opportunistically exploits storage and communications to be disruption tolerant and increase availability of data in the network. Initial evaluations of DisService in the context of data harvesting and dissemination of SA data show promising results.

Future planned enhancements to DisService include optimizing the replication and forwarding strategies to explicitly support multiple dissemination patterns (any combination of one|few|many publishers to one|few|many subscribers) and adding supports for predicates to be attached to subscriptions. DisService is also being extended to incorporate learning mechanisms to predict information needs of users in order to pre-stage the information.

8. ACKNOWLEDGEMENTS

This work is supported in part by the U.S. Army Research Laboratory under Cooperative Agreement W911NF-04-2-0013, by the U.S. Army Research Laboratory under the Collaborative Technology Alliance Program, Cooperative Agreement DAAD19-01-2-0009, and by the Office of Naval Research under grant N00014-09-1-0012.

9. REFERENCES

[1] Suri N., Rebeschini M., Breedy M., Carvalho M., and Arguedas M. Resource and Service Discovery in Wireless Ad-Hoc Networks with Agile Computing. In Proceedings of the 2006 IEEE Military Communications Conference (MILCOM 2006), October 2006, Washington D.C.

[2] Tortonesi M., Stefanelli C., Suri N., Arguedas M., and Breedy M. Mockets: A Novel Message-oriented Communication Middleware for the Wireless Internet, in Proceedings of International Conference on Wireless Information Networks and Systems (WINSYS 2006), Setùbal, Portugal, August 2006.

[3] Carvalho M., Suri N., Arguedas M. (2005) Mobile Agent-based Communications Middleware for data Streaming in the Battle field. In Proceedings of the 2005 IEEE Military Communications Conference (MILCOM 2005), October 2005, Atlantic City, New Jersey.

[4] Suri N., Marcon M., Quitadamo R., Rebeschini M., Arguedas M., Stabellini S., Tortonesi M., Stefanelli C. An Adaptive and Efficient Peer-to-Peer Service-oriented Architecture for MANET Environments with Agile Computing. In Proceedings of the second IEEE Workshop on Autonomic Computing and Network Management (ACNM'08).

[5] SQLite Relational Database Library. Online reference: http://www.hwaci.com/sw/sqlite/.

[6] Quinlan R. C4.5: Programs for Machine Learning. Morgan Kaufmann Publishers Inc., 2004.

[7] Nlake C., Keogh E., and Merrz C. UCI Repository of Machine Learning, 2007. Online reference: http://archive.ics.uci.edu/ml/.

[8] Carson M., Santay D. NIST Net – A Linux-based Network Emulation Tool. National Institute of Standars and Technology.