DDAM: Dynamic Network Condition Detection and Communication Adaptation in Tactical Edge Networks

Roberto Fronteddu[1], Alessandro Morelli[2], Mauro Tortonesi[2], Niranjan Suri[1,3], Cesare Stefanelli[2], Rita Lenzi[1], Enrico Casini[1]
[1]Florida Institute for Human & Machine Cognition (IHMC), Pensacola, FL USA
[2]Department of Engineering, University of Ferrara, Ferrara, Italy
[3]U.S. Army Research Laboratory (ARL), Adelphi, MD USA
{rfronteddu, nsuri, rlenzi, ecasini}@ihmc.us
{alessandro.morelli, mauro.tortonesi, cesare.stefanelli}@unife.it
{niranjan.suri.civ}@mail.mil

***Abstract*—Tactical Edge Networks provide one of the most challenging communication environments. In order to cope with node mobility, constrained resources, and link unreliability, communication solutions designed for Tactical Edge Networks typically present highly configurable interfaces to be adaptable for various networking conditions. However, the extreme dynamicity and heterogeneity of tactical scenarios call for network-aware, adaptive communication systems that continuously re-tune their configuration parameters to match the ever-changing network conditions. This paper presents the Dynamic Detect and Adapt Mechanism (DDAM) of the Agile Communication Middleware, a distributed solution to perform network monitoring and communication adaptation specifically designed for Tactical Edge Networks. The present work focuses on two components of the DDAM: NetSensor, which provides efficient monitoring of the network status, and NetSupervisor, which is responsible for characterizing the network technology used to connect a pair of nodes in the network. The presented results show that our solution can accurately identify the technology used to establish links between nodes.**

***Index Terms*— Computer network management, Distributed computing, Data analysis, Middleware, Mobile ad hoc networks.**

I. Introduction

Tactical Edge Networks (TENs) are one of the most challenging communication environments. Frequent node mobility, resource constrained devices, a wide range of heterogeneous wireless technologies, and strongly varying (and sometimes access denied) environmental conditions mean that distributed applications operating in TENs need to cope with frequent disconnections and fluctuating radio channel conditions. Consequently, TENs cannot operate with traditional communication solutions, but need purposely-designed communications middleware.

Communication solutions designed for TENs are typically highly configurable and can be tuned to work under different conditions. However, the extreme dynamicity and heterogeneity of TENs make it impossible to optimally tune the behavior of communications middleware using predefined configurations or simple heuristic solutions. Network aware solutions, capable of continuously re-tuning the configuration parameters of communications middleware to adapt its behavior to the ever changing operating conditions are needed. However, accurately detecting the current network conditions and reacting accordingly still represents an open research question.

Dedicated solutions are needed that are capable of continuously monitoring the state of nodes and communication links in TENs to detect the current network conditions, of analyzing the collected data to build an actionable and in-depth knowledge of the current state of the network (including load, link type, quality, and network topology), and of providing this information to communication solutions.

This paper presents Dynamic Detect and Adapt Mechanism (DDAM), a comprehensive solution for network monitoring and communication adaptation specifically designed for TENs. DDAM integrates with the comprehensive communication solutions provided by the Agile Communications Middleware (ACM) to implement adaptive and network aware communications for TEN environments [1] [2] [3].

II. Network Awareness in Tactical Networks

The extreme heterogeneity and dynamic nature of TENs present an interesting challenge from a communications management and optimization perspective. The process of determining network conditions and adapting the communications strategies accordingly is a difficult problem, further exacerbated by the constrained and variable nature of bandwidth, computational, and storage resources in TENs. Additional complications emerge from the communication patterns of modern sophisticated multi-party applications,

which use many traffic streams with very different characteristics and needs. For example, commonly provided services range over time from low-level diagnostic updates to high-level video streams. These frequent variations exhibited by both applications and networks can lead to different traffic patterns, and potentially to difficult troubleshooting challenges.

Network performance has a direct impact on the timely delivery of information to nodes, which could be critical for situation awareness and mission success. Network awareness and a smart decision process can help address this challenge. Baseline measurements of the network and the ability to determine which applications and devices are generating anomalous amounts of traffic are useful tools that network architects can use to build adaptive networks.

Active and passive network monitoring are essential and critical parts of a network-aware system. Active monitoring entails injecting test traffic onto a network to monitor the flow of that traffic, whereas passive monitoring is based on the observation of traffic that is already traversing the network. Passive monitoring requires a device on the network to capture network packets for analysis. Some common options are to use specialized probes designed to capture network traffic or the integration of built-in capabilities on switches or other network devices. By injecting traffic, active monitoring has the drawback of changing the normal state of the network and adding overhead, thus compromising the accuracy of the acquired data. This overhead adds to the traffic already needed to convey network statistics to network analysis systems, and to the traffic needed to notify applications that require updates about how to improve their use of network resources. This overhead is particularly relevant in TENs where performance and bandwidth vary greatly and can be very constrained to start. On the other hand, the information acquirable via passive monitoring is limited to active end-to-end communications.

Another relevant problem in network administration and monitoring techniques, applied to TENs, is the communication security (COMSEC) barrier that encrypts packets end-to-end across tactical radio networks, and blocks signaling of network state from the radio network to the protected platform elements [4]. Furthermore, there is also the inability of components, operating in protected subnetworks, to detect the types of radios, waveforms, and configuration used by the underlying network. Significant prior work has focused on how to use monitored statistics such as congestion level, packet loss rate, and available path capacity in order to estimate metrics that affect performance of applications. However, proposed solutions generally do not address the problem of determining which underlying communication technologies are being used, and assume to already have notions about it, or ignore the problem altogether [4] [5]. Each of the many communication technologies deployed in TENs has specific and peculiar characteristics that network administration systems and adaptive applications cannot ignore if their purpose is to use networks as efficiently as possible.

Finally, another matter of concern is how to use the harvested data and subsequent deductions to plan countermeasures and best courses of action to better adapt to network status and changes. The very constrained nature of TENs greatly reduces the size of traffic that monitor systems can use for control, without risking to compound the congestion problem during times of low network performance. Therefore, it is important to choose the number and type of metrics to distribute carefully. These same constraints make the overhead introduced by polling-based mechanisms extremely unappealing.

III. DDAM

To address the problems described in the previous section, we designed the Dynamic Detect and Adapt Mechanism (DDAM), a distributed and extendable middleware solution capable of harvesting, analyzing, and sharing, traffic statistics and analysis results over the network. DDAM has the objective of improving the use of available network resources and of integrating with the components of the Agile Communications Middleware by dynamically adapting their behavior to the detected network status. An example of the deployment of DDAM components in a tactical network is given in Fig. 1.

DDAM has four main components and a GUI based visualization tool. NetSensor (NSens) is a lightweight network monitoring solution capable of passively harvesting a variety of network traffic information. Node Monitor (NMon) is in charge of cleverly fusing network statistics collected by various ACM components, including NetSens, and of distributing them to other NMon and NetSupervisor (NSup) components. NSup is a high-level network analysis software capable of analyzing network statistics and other information provided by NMon to infer the type and quality of network links between any couple of nodes. NSup shares its knowledge of the current network conditions with a local NMon, which will in turn disseminates it to other remote Node Monitors, along with any other collected information. Information replication across the system allows for coping with the network partitioning phenomena caused by the severe node mobility. Adaptive Communications Management System (ACMS) receives information from NMon about the types of link detected by NSup and executes policy-based decisions concerning the adaptation of the traffic generated by ACM components running on the nodes. Finally, NetViewer (NView) is a visualization utility that can subscribe to a NMon instance to present the information about current network status and topology collected by DDAM in graphical form to human observers.

We designed the components of DDAM with the target of providing a distributed solution to enable network-aware adaptivity within the ACM. The challenges of TENs drove our design choices concerning the tasks that each component carries out, the way replication and distribution of information has been handled, and the lack of any centralized component.

A. NetSensor

NSens is a monitoring agent able to collect information about the status of the network. Monitoring in NSens is purely passive

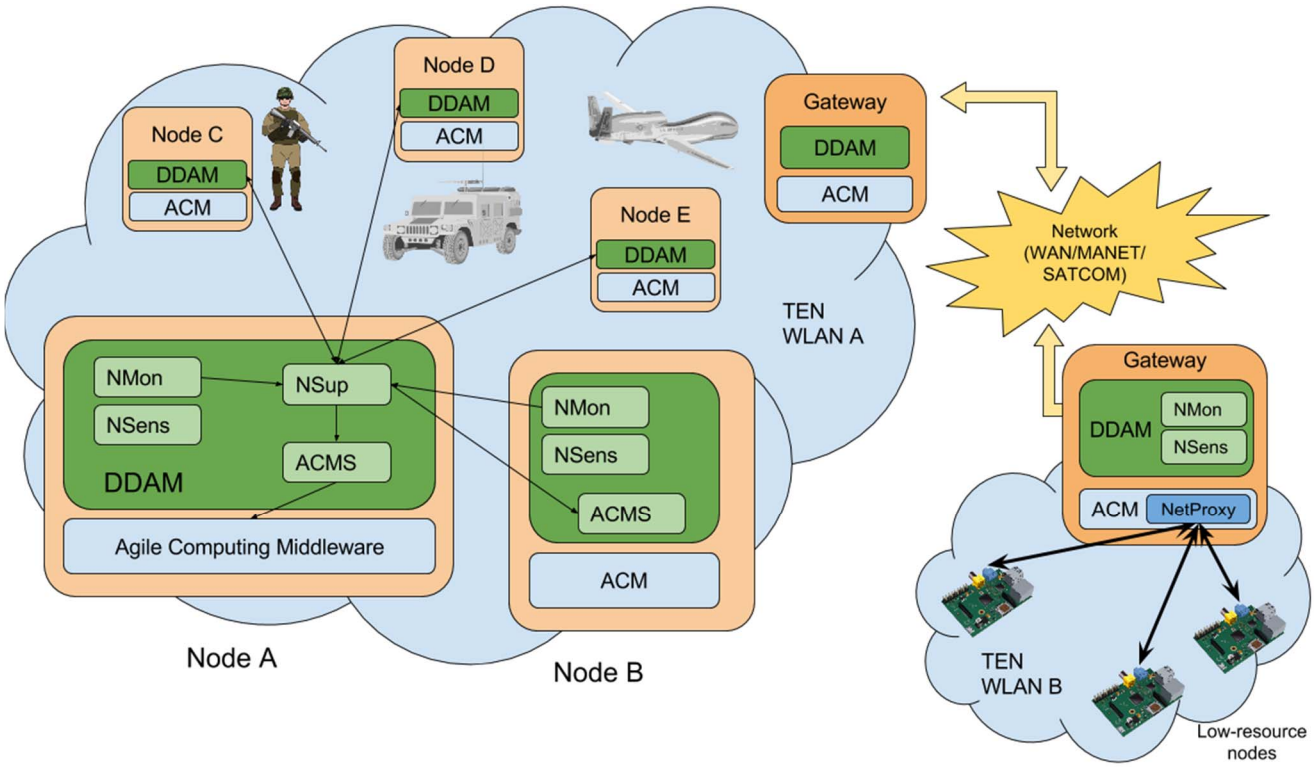

Figure 1 – Example of deployment of DDAM components over the nodes of a TEN

and relies on the Pcap/WinPcap[1] library. Its main purpose is to extract traffic information and build statistics based on Source/Destination IPs, MAC addresses, protocol and port number, to monitor network usage, to discover and identify any local gateways, and finally to assess the latency between any pair of nodes by passively harvesting ICMP or (S)ACK/NAK packets. We tailored the information that NSens collects specifically to face the problem of identifying the technology used to connect a pair of remote nodes.

After the sensing and aggregation phase, NSens encodes the information in Protobuf packets [6], format that presents interesting encoding and efficiency performances, and sends them to an instance of NMon, which will make the data available to other nodes and components. If a NMon is available locally, NSens will send the harvested statistics to the local component; otherwise, they will be sent to a remote instance. As in the example in Figure 1, placing NSens very close to (or embedded in) a local network gateway is favorable. By doing this, NSens would be able to harvest all the data exchanged by local and remote nodes without the need to install a component on each node of the local network. This enables the analysis of network traffic produced by resource constrained nodes, without having those nodes running NSens themselves.

Another interesting feature of NSens is its local topology detection mechanism, which works as described in the following two algorithms.

1) Gateway Identification algorithm

Input: Packet p.

Output: A list containing the detected gateways.

1. Save MAC and Source IP address in the MacIp table;
2. **If** p is an arp reply packet **then**
3. Store the MAC and IP source addresses of the packet in the arpMacIp table;
4. **If** Exists a MAC entry in the MacIp table with more than five IPs associated **then**
5. Add that MAC to the possibleGateway list;
6. **If** Exists match m between an arpMacIp table's entry and a possibleGateway list's entry **then**
7. Save match m in the gateway List.

2) Local Nodes Identification algorithm

Input: Packet p.

Output: Characterization of the packet source address.

1. **If** p Source MAC is contained in the gateway List **and** p Source IP is not a Gateway IP **then**
2. p source is not a local node;
3. **Else If** p Source IP is a Gateway IP **then**
4. p source is a local Gateway;
5. **Else**
6. p source is a local node.

NSens was designed to operate either as a stand-alone component or integrated with NetProxy operating in Gateway Mode [3], in order to reduce the computational overhead on the node.

B. NodeMonitor

NMon is a distributed component able to collect, aggregate, and share network data from multiple sources, e.g., NSens agents and other ACM components, such as Mockets, which

[1] Available online at www.tcpdump.org

can provide accurate latency information on end-to-end connections by means of active probing, and NetProxy. NMon also directly senses a variety of information from the local node, including CPU, memory, battery, and OS-related information. These data are summarized and shared following a peer-to-peer approach with other NMon instances, in order to distribute the information over the whole network. The dissemination mechanism is based on an enhanced version of the Group Manager [7], which exploits multicast traffic to reduce the load on the network.

Another interesting feature is the ability of NMon to produce a representation of the global topology of the network by merging local topology information harvested by NSens agents. NMon can then share this topology data with other components such as NSup for analysis, or with the NView component to provide a graphical representation of the network topology.

The smart update mechanism incorporates a few different strategies to reduce the control overhead generated over the network. Updates are typically limited to 1 MTU-sized packet and the update rate is adjusted dynamically based on the observed network capacity. Since the data available locally usually exceeds 1 MTU, subsets of the data are selected based on freshness and change rate. However, data that is skipped has its priority raised so that it will not be left out indefinitely.

C. NetSupervisor

NSup is a high level Network Analyzer. This component's task is to receive merged data from an instance of NMon, analyze it, and make deductions about the status of the network and the links. NSup analyzes the aggregated information provided by NMon and returns, for each couple of nodes A and B, two tuples containing four fields each: Link Type, Throughput, Packet Loss, and Latency. Link Type represents the communication technology used to connect the two nodes, which can be one among LAN, SATCOM, or HF.

NSup computes Throughput from the packets received by Node B and sent by Node A and latency from the time packets need to get from Node A to Node B. The component also calculates Packet Loss using (1), where ABOData is the data sent by node A to Node B, and BAIData is the data received by Node B and sent by Node A.

$$PL = (ABOData - BAIData) * 100 / ABOData \quad (1)$$

To assess the link type, NSup implements a deterministic selection algorithm based on the specific characteristics each link type is supposed to have. NSup loads data from a configuration file that specifies the typical values of throughput and latency for each Link Type, and creates a reference table for each one. In addition, the configuration file also specifies a score for each entry of the Reference Table. Table I shows an example of a set of expected values for three link types, whereas Table II shows the structure of the Reference Table that will be created for each link type specified in the configuration file. The score specified for each entry reflects how likely a monitored characteristic compares to the typical one for a link, with higher scores indicating higher plausibility. In Table II, ET and EL are the expected throughput and latency specified in the Connection Table, whereas T and L are the run-time values obtained by monitoring the network.

During the Link Analysis phase, NSup will use the reference values specified in the configuration file to assign a score to each configured link type. Each score will be the sum of the sub-scores obtained for latency and throughput values, each one extrapolated in turn by comparing the value obtained at run time with the associated reference table mentioned earlier. Finally, NSup will select the link type with the higher score as the assumed link type. For example, if we have two link types named A and B, and at run time we detect a throughput of value T and latency of value L, the score S of A can be obtained by using (2), whereas the score S of B by using (3). Sba and Sla are functions that associates a score values to the throughput T and latency L by looking respectively at the throughput and latency scores tables associated to A. The same reasoning applies for Sbb and Slb, which are the scores function associated to B.

$$S(A) = Sba(T) + Sla(L) \quad (2)$$

$$S(B) = Sbb(T) + Slb(L) \quad (3)$$

The scores can also be evaluated as a confidence index, the larger the difference between each score value, the safer the classification of the link. For the sake of the argument, if after the extrapolation the total score of A is higher than the total score of B, NSup will infer that the link type is A and vice versa. Table II shows all the thresholds for which a score has to be defined.

TABLE I – EXEMPLAR VALUES FOR A CONNECTION TABLE

	Bandwidth	Latency
WLAN	<100Mbps	<=10ms
SATCOM	<1Mbps	>=200ms
HF	<56Kbps	<=100ms

TABLE II – REFERENCE TABLE

Throughput Score table	Latency Score table
T >= ET	0.8 * EL <= L < 1.2 * EL
0.9 * ET <= T < ET	1.2 * EL <= L < 0.2 * EL
0.75 * ET <= T < ET	L > 2* EL
0.50 * ET <= T < ET	0.5 * EL <= L < 0.8 * EL
0.25 * ET <= T < ET	0.2 * EL <= L <0.5 * EL
0.1 * ET <= T < ET	0.1 * EL <= L < 0.2 * EL
T < 0.1 ET	L < 0.1 * EL

D. Adaptive Communications Management System

The ACMS is the component of the DDAM system that performs the smart adaptation of the traffic produced by other ACM components running in the network. The ACMS implements a decision-making logic that processes the information produced by NMon and NSup concerning the network status, its topology, and the type of links that connect the nodes in the network. The output of this process is a network traffic management plan that aims at adapting the behavior of (a part of) the ACM components deployed in the network to optimize the usage of the scarce resources available in the system.

The final target of the ACMS is to improve the system performance by taking advantage of new resources that enters the network temporarily, for instance due to mobility of nodes, or increasing the amount of low priority traffic that can enter the network when links are underutilized or their quality improves. At the same time, the ACMS ensures that the system can provide the necessary QoS to the most critical applications even when the network is overloaded.

We designed the ACMS to interact fully with other components of the ACM, such as Mockets [1], DisService [2], and NetProxy [3] (details are available in the cited works). This enables the ACMS to have almost complete control over the network traffic, by directly affecting the priority of flows or the packet delivery semantics used, enabling or disabling data compression for specific connections, forwarding traffic over new links, and so forth.

Due to space limitations, the experimental results and the algorithms presented in this paper will only focus on NSens and NSup.

IV. Experimental Results

To demonstrate the reliability of the NSup algorithm to detect link types in TENs, we present the results obtained in a lab configuration designed to reproduce common TENs pattern in link type characterization. To this avail, we ran the experiments in an emulated environment, which allowed us to reproduce the characteristics of a TEN. More specifically, we used the Mobile Ad-hoc Network Emulator (MANE) [8], to set up connectivity between the nodes involved in testing.

The nodes are part of the NOMADS testbed, which comprises 96 HP DL140 servers connected through a 100Mbps Ethernet LAN.

For the sake of experimentation, we modelled three types of link with MANE, summarized in Table I. In addition, we applied a 20% random noise to the statistics harvested by NSens, in order to test the stability of the NSup scoring process. The experiment consisted of randomly variating the traffic output and the link type of the link connecting two sub-networks in a configuration similar to the one shown in Fig.1. Table III and table IV summarize the score tables used during the tests.

The graphs in Fig. 3-5 show the score level in different tests. We performed tests using a LAN link, a SATCOM link, and a HF link. NSup was correctly able to identify the link type 8 times out of 9 by a relevant margin from the scores of other link types.

The noise caused the third test done with the HF link to fail. NSup was in fact unable to identify an outperforming HF. Still the likeliness of the inferred scores gives an indication of the anomalous condition. In all the other experiments, NSup identified the link types correctly.

Table III – Throughput Scores used for the experiments

Throughput Scores table	LAN	SATCOM	HF
T >= ET	100	-20	-20
0.9 * ET <= T < ET	90	90	90
0.75 * ET <= T < ET	90	100	100
0.50 * ET <= T < ET	80	80	100
0.25 * ET <= T < ET	70	70	100
0.1 * ET <= T < ET	60	60	90
T < 0.1 ET	40	40	80

Table IV - Latency Scores used for the experiments

Latency Scores table	LAN	SATCOM	HF
0.8 * EL <= L < 1.2 * EL	100	100	100
1.2 * EL <= L < 2 * EL	100	100	100
L >= 2 * EL	50	100	60
0.5 * EL <= L < 0.8 * EL	90	40	70
0.2 * EL <= L <0.5 * EL	90	30	40
0.1 * EL <= L < 0.2 * EL	90	20	30
L < 0.1*EL	90	10	20

V. Related Work

Many solutions have been proposed in the field of network administration and resource monitoring applied to TENs. More often than not, these solutions use the SNMP protocol in some way or another. In [9], SNMP performance in TENs are evaluated and the results show that SNMP was designed with high performance networks in mind. For example, each SNMP packet contains many fields such as version, community ID and an OID for each object in the variable-binding list, where OIDs sometimes requiring more space than the data itself. Our approach avoids this problem by coding information using Google Protocol Buffers (Protobuf) [6]. Fig. 2 shows that the relation between serialization overhead growth and number of entries in the Protocol Buffer message is less than linear.

Another problem in using SNMP, as explained in [9], is that SNMP does not support Multicast with evident overhead implications in the polling phases. Our solution instead makes use of a Multicast Group for the delivery of update containing statistics and other information, thus reducing the overhead. We essentially perform a local broadcast by configuring a TTL of 1, but that could be increased, thus increasing the awareness in the network, at the cost of more bandwidth usage. In addition, unlike a polling process, NMon instances already harvest all the information available in the local sub-network and then share only a summarized version with other nodes. The verbosity of this information can change based on the monitored characteristics of the link connecting the local sub-network with other sub-networks.

Two systems similar to DDAM are described in [4] and [5], the first implementing its own monitoring system, the second using SNMP packets. In [4], the authors suggest to deploy the passive monitoring system close to the COMSEC gateway, a decision that we also made as shown in Fig.1. One difference is the way the network metrics are shared.

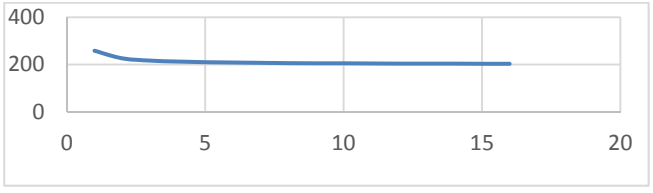

Figure 2 – Ratio between packet size in Bytes and number of entries in the serialized Protocol Buffer message.

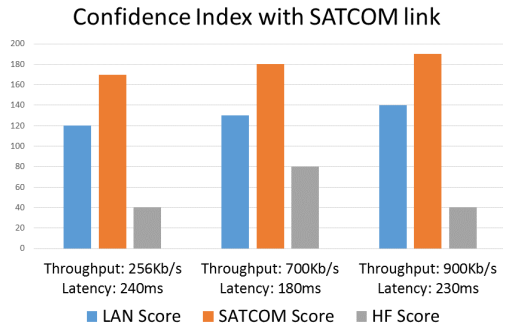

Figure 3 - SATCOM link score results.

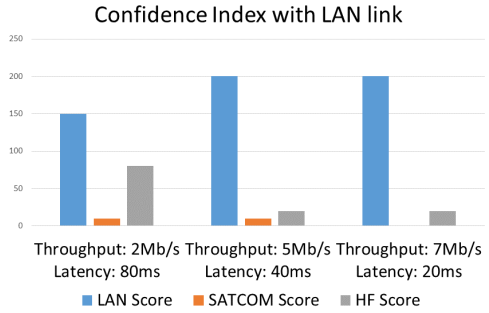

Figure 4 - LAN link score results.

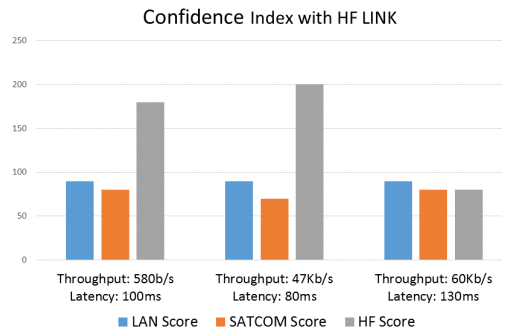

Figure 5 – HF link score results.

The solution presented in [4] shares this information for a per flow metric, where each flow is characterized by source/destination, port, IP protocol, and priority, instead of being summarized and consolidated with other metrics before being sent as in our solution. This gives DDAM an advantage in term of overhead, number of packets sent, and optimization to fit in one MTU.

Another difference between our solution and the one presented in [4] is that that solution uses the harvested information to detect network congestion. Since different transmission technologies have different characteristics, we are inclined to think that by using our approach the congestion detection mechanism could be further refined by first identifying the link technology, for example by leveraging information about the expected packet loss, to better tailor the threshold used by their algorithm.

The performance monitor of the solution presented in [5] instead uses Network Management Station (NMS), which in turn uses SNMP to retrieve network statistics. Thus, the solution is affected by the aforementioned SNMP problems. In addition, their solution may not be viable when COMSEC is present, since the security layer would block SNMP notification for the reasons explained in section 2.

The solution presented in [5] also implements an interesting approach to the network-resource allocation problem where a centralized component is responsible for allocating network resources, whereas we envision that the results provided by NSup will be shared with other components, which will in turn adapt to the network conditions in an independent way.

VI. Conclusions

A primary challenge in TENs is represented by the necessity of designing applications that can dynamically adapt to network and connection changes. One big challenge to achieving this is the obscurity introduced by the communication security barrier (COMSEC), which blocks signaling of network state from the radio network to the protected platform elements. This results in applications running behind the security barrier to be unaware of the wireless connection where capacity-constrained links reside. To this avail, we developed DDAM, a distributed solution able to infer links type and status behind the security wall. DDAM has the potential to enhance the utility of tactical applications by enabling them to adapt their behavior based on the detected status of the links, for example by reducing the output of low priority traffic when NSup detects a particularly constrained link. DDAM also simplifies the task for algorithms or other components to be able to estimate network congestion, or other network anomalies, by providing knowledge about the type of link under observation.

References

[1] N. Suri, E. Benvegnù, M. Tortonesi, C.Stefanelli, J. Kovach, J. Hanna, "Communications Middleware for Tactical Environments: Observations, Experiences, and Lessons Learned", IEEE Communications Magazine, ISSN 0163-6804, Vol. 47, No. 10 (Special Issue on Military Communications), pp. 56-63, October 2009.

[2] N. Suri, G. Benincasa, M. Tortonesi, C.Stefanelli, J. Kovach, R. Winkler, R. Kohler, J. Hanna, L. Pochet, S. Watson, "Peer-to-Peer Communications for Tactical Environments: Observations, Requirements, and Experiences", IEEE Communications Magazine, ISSN 0163-6804, Vol. 48, No. 10 (Special Issue on Military Communications), pp. 60-69, October 2010.

[3] M. Tortonesi, A. Morelli, C. Stefanelli, R. Kohler, N. Suri, S. Watson, "Enabling the Deployment of COTS Applications in Tactical Edge Networks", IEEE Communications Magazine, ISSN 0163-6804, Vol. 51, No. 10 (Special Issue on Military Communications), pp. 66-73, October 2013.

[4] T. Chen, S. Eswaran, M. A. Kaplan. S. Samtani, D. Shur, J. Sucec and L. Wong, “Enhancing Application Performance with Network Awareness in Tactical Networks”, in Proceedings of IEEE Milcom 2011.

[5] A. S. Peng, D. M. Moen, T. He, and D. J. Lija, “Automatic Dynamic Resource Management Architecture in tactical Network Environments” arker, Proc. IEEE Milcom 2009.

[6] Google Protocol Buffer, available online at: https://developers.google.com/protocol-buffers/docs/encoding

[7] N. Suri, M. Rebeschini, M. Breedy, M. Carvalho, and M. Arguedas, “Resource and service discovery in wireless AD-HOC networks with Agile Computing”, in Proceedings of IEEE Milcom 2006.

[8] Mobile Ad-Hoc Network Emulator (MANE), available online at: http://cs.itd.nrl.navy.mil/work/mane/index.php

[9] G. Kuthethoor, P. Sesha, J. Strohm, P. O’Neal, G. Hadynski, D. Climek, J. DelMedico, D. Kiwior, D. Dunbrack and D. Parker, “Performance analysis of SNMP in Airborne Tactical Networks.” Proc. IEEE Milcom 2008.