NETWORK CONDITIONS MONITORING IN THE MOCKETS COMMUNICATIONS FRAMEWORK

Cesare Stefanelli, Mauro Tortonesi
ENgineering Department In Ferrara (ENDIF)
University of Ferrara
Ferrara, Italy

and

Marco Carvalho, Niranjan Suri
Florida Institute for Human & Machine Cognition
Pensacola, FL, USA

ABSTRACT

Communication between mobile devices in the MANET scenario exhibits significant reliability and performance problems. Traditional communication infrastructures designed for wired networks are not well suited for MANETs because they masquerade network level conditions. In the MANET scenario, instead, there is the need to expose current network conditions to applications, enabling them to adapt their behavior to changes in the quality of the communication links. In the context of a MANET-oriented communication framework (called Mockets), this paper presents a network conditions monitoring component which provides applications with timely and accurate information about communication channel characteristics. In particular, the paper focuses on the measurement of latency, obtained via Round Trip Time measurement, for which it presents three different algorithms. The experimental results show the performances of the different algorithms in a MANET-like emulation environment.

I. INTRODUCTION

In the MANET scenario, mobile devices can communicate and coordinate wirelessly without pre-installed infrastructures. Mobile nodes can move and can arbitrarily join or leave the network, dynamically changing the topology, the set of available resources, and the information sources. Therefore, communications in MANETs are subject to highly variable conditions and exhibit significant reliability and performance problems. Network-Centric Warfare often involves systems of this nature.

Traditional communication infrastructures and protocols were designed for wired networks and steady-state communication channel conditions. In addition, they provide applications with a network transparent programming model, which completely masquerades the dynamicity of the communication channel. In those infrastructures, applications can not detect changes in channel status and can not adapt to highly varying network conditions [1].

For instance, the Internet Protocol Suite is often proposed as the foundation for communications in MANETs because of the significant benefits it provides in terms of standardization, wide equipment choices, and low setup and operations costs [2]. However, the TCP/IP programming model is not well suited for MANETs as it does not provide support for mobility and does not allow any applications-driven QoS adaptation decisions, e.g, by imposing a constraint on the number of packet retransmissions in order to decrease the latency in delivery of enqueued information.

Today, it is generally accepted that the performance and robustness of distributed applications in MANET environments can be enhanced through network-aware programming models. These models provide applications with the information about the dynamically changing network conditions such as round-trip time and available bandwidth of the communication link. Depending on the current network conditions and the nature of the information being exchanged, applications can adapt their behavior dynamically by exploiting QoS adaptation mechanisms [3].

The Mockets framework is an example of a communications middleware specifically designed to address the challenges of MANET scenarios [4], [5]. Mockets is an application-level library that resides on top of the operating system and communication protocol stack. This favors easy deployment, platform independence, and portability. Mockets supports terminal mobility and permits migration of communication endpoints (Mockets stands for mobile-sockets). In addition, Mockets adopts a network-aware approach and exposes the dynamically changing network conditions to applications that can thus adapt their behavior accordingly. In this way, network-aware Mockets applications can perform continuous service provisioning in spite of abrupt changes in current network conditions. This capability is important to enable the development of robust, adaptive, and efficient distributed applications in the MANET environment.

In the context of the Mockets framework, this paper presents the network conditions monitoring component, which provides applications with timely and accurate information about varying communication channel

1-4244-1513-06/07/$25.00 (c)2007 IEEE

characteristics. For every established connection, the Mockets monitoring component continuously measures network level parameters, such as available bandwidth along the communication path, packet loss rate, round-trip time and peer reachability. In particular, the paper focuses on the measurement of network latency, obtained via Round Trip Time measurement, for which it proposes three algorithms: *ACK-based*, *timestamp-based*, and *timestamp-based with compensation*. The algorithms have different characteristics in terms of agility, stability, and computational overhead. The experimental results presented in the paper provide a performance comparison of the algorithms in two different emulation environments configured to reproduce MANET-like networking conditions.

II. ADAPTIVE APPLICATIONS IN MANETS

In a MANET environment, network conditions may be turbulent and chaotic. Node mobility, churn, and unreliability of wireless communications cause major fluctuations in the network resources available to applications. As a result, applications deployed in MANETs need to withstand abrupt changes in network performance and to dynamically adapt their QoS requirements accordingly.

Traditional communication protocols provide applications with an inappropriate programming model that hinders their QoS adaptation process. In fact, by masking underlying network conditions, the traditional network programming model denies applications any feedback about network performance. This effectively prevents them from making informed and timely QoS adaptation decisions.

There is a need for network-aware programming models which enable the development of adaptive applications on MANETs. This requires the introduction of communication middleware that continuously monitors network performance and processes collected data to extrapolate network statistics that are then provided to applications according to their specific interests. Timely and accurate network condition monitoring is of crucial importance to allow QoS adaptation [3].

Applications could then use the information about network conditions provided by the middleware to perform prompt and informed QoS adaptation decisions. In particular, applications could modulate their QoS requirements in the case of changes in the quality of the communication links. For instance, applications could decide to scale their service level by modifying their service logic. In the case of video, this adaptation might include changing the frequency, resolution, or color depth of the video frames they exchange. Alternatively, applications can improve their resilience to channel unreliability by exploiting different communication mechanisms such as redundant retransmissions, resizing of network buffers, modulation of transmission priorities, or partial reliability mechanisms.

III. THE MOCKETS FRAMEWORK

The Mockets framework is an application-level communications library that has been designed to support adaptive applications in MANET environments. Mockets provides a number of unique features including multiple service classes (with options for reliability and sequencing), prioritization, message tagging and replacement, detailed communication statistics, numerous timeout options, policy-based bandwidth control, and endpoint migration.

The Mockets framework has been realized as a library that resides at the application level and not as part of the operating system kernel or network protocol stack. This was an important design choice in order to support easy deployment, platform independence, and phased utilization. By being operating system and network stack agnostic, the Mockets framework can be used in any operating system environment and is currently available for Win32, Linux, and MacOSX platforms. Also, being an application library allows a subset of the applications to use Mockets while other applications can continue to use TCP and UDP. This facilitates deployment by allowing applications to be gradually migrated to use Mockets instead of the sockets API.

Mockets is extensively used in several research projects, such as the Agile Computing Framework for opportunistic resource discovery and exploitation in extremely dynamic environments [6]. In addition, the Mockets communication library has been successfully used by the Army Research Laboratory as part of the Warrior's Edge initiative of the Horizontal Fusion Portfolio's Quantum Leap demonstrations and in the C4ISR On The Move exercise. Finally, the library is also currently being used by the Air Force Research Laboratory in internal experimentation and evaluation. More detailed information about Mockets, including a throughput comparison with TCP, is available in [4], [5], [7].

Mockets provides applications with several fine-grained QoS adaptation mechanisms, leaving them the task of implementing appropriate adaptation strategies and policies. The rationale behind this strict separation of concerns is that only applications have the required

knowledge to perform decisions about adaptation, according to service logic and user preferences.

QoS adaptation mechanisms provided by Mockets include adjustment of sender and receiver buffer, aggressive retransmissions, partial reliability, transmission priority modulation, and cancellation and/or replacement of obsolete information in the transmission queue. The wide range of QoS adaptation mechanisms provided by the Mockets middleware allow applications to choose the best suited service adaptation strategy depending on application logic, current network conditions, and user preferences.

To support applications in performing prompt and smart QoS adaptation decisions, Mockets provides them with timely and accurate information about network conditions. In particular, Mockets gathers and exports a number of parameters: time since last contact, bytes sent, packets sent, packets retransmitted, bytes received, packets received, packets discarded (three different categories), estimated RTT, pending data size, pending queue size, unacknowledged data size and unacknowledged queue size. In addition, counts of messages sent (overall and per message type) are also available. The rest of the paper presents in detail the network conditions monitoring function of Mockets, with a special emphasis on network latency estimation.

As an example of Mockets-based application, let us consider a disaster recovery scenario where a human operator commands a robot working in hazardous environment. The remote control application needs to deal with time-sensitive data (such as a video feed from a camera installed on the robot and periodic update messages from the robot sensors) and critical data (such as movement commands from the operator). As a result, it can decide to exploit different delivery services, among the many provided by the Mockets middleware, for the two classes of data. For instance, the application could use sequential unreliable delivery for sensor updates and video frames, and reliable delivery with aggressive retransmissions for commands from the operator. This would ensure the delivery and processing of critical data with the utmost precedence, and guarantee the correct operation of the robot in emergency situations. In addition, Mockets monitors network conditions and can notify the application in the case of a communication channel quality falling below a given threshold. The application can thus downscale its service level by reducing the quality or frequency of video frames or sensor updates.

IV. NETWORK CONDITIONS MONITORING IN MOCKETS

The Mockets middleware monitors network conditions and provides applications with timely and accurate information about the underlying communication channels characteristics. In particular, for every established connection, Mockets continuously monitors the channel status along the communication path, performing both estimation of network level parameters and storage/processing of obtained data at the end points. Mockets adopts the distributed end-to-end approach because that is better suited to the MANET environment than centralized solutions. In fact, the topology of MANETs is extremely dynamic, thus preventing the storage and processing of network monitoring information on server nodes along the communication path. In addition, the election of server nodes executing network monitoring tasks on account of other nodes raises both fairness (e.g., server nodes would consume their batteries faster) and security concerns (e.g., server nodes might maliciously provide false information).

The Mockets middleware implements a hybrid network monitoring scheme. In fact, Mockets performs both active measures by means of packet probing (e.g., for peer reachability and bandwidth measurement) and passive measures (e.g., for collecting statistics about connections). This dual scheme combines the benefits of both active monitoring, in terms of agile and broad spectrum measures, and passive monitoring, in terms of low network overhead. As a result, the hybrid approach is particularly well suited for highly dynamic networking environments in which wireless communications take place [3], [8].

The Mockets middleware is also capable of leveraging information provided by lower-level protocols. In such cases, end-to-end metrics are augmented with local information about link quality and MAC retransmissions, as well as details about local topology and contention graph estimates. One example of such an application is the integrated version of Mockets and IHMC's Xlayer communications substrate [9], which currently provides synchronized aggregate logging capabilities across multiple layers, mapping Mockets data flows with actual MAC packet transmissions. The same capabilities can be extended to support a feedback loop between Mockets and lower level protocols (MAC and PHY) to enable application-driven MAC scheduling and topology management.

Mockets-based applications can directly query the middleware for the current value of the desired network level parameters. For maximum flexibility, applications can also adopt a publish-subscribe model to register their interest in a specific set of parameters. Mockets will then

notify applications when the registered parameters change via a callback interface.

In the Mockets architecture, as depicted in Figure 1, the Network Conditions Monitor (NCM) is the component that monitors network status, processes the collected information and provides it to applications. NCM interfaces with the Receiver component to collect information about network conditions from the underlying layer, and with the Transmitter component to schedule the transmission of active probes in the network.

In particular, the NCM component detects peer unreachability via a keep-alive mechanism, which allows quick discovery of problems at the link and network layers. For instance, NCM can detect changes in the network layer address of a device and can cooperate with the Session Management component to support terminal mobility (by enabling automatic rebinding of established connections without forcing applications to shut down and reinstate all their network connections).

In addition, the NCM component performs bandwidth estimation by processing information obtained from the XLayer substrate and by actively probing the network using packet dispersion mechanisms [10]. NCM also passively monitors the status of established connections, collecting statistics such as packet loss rate, number of retransmitted packets, etc. Finally, NCM performs latency estimation by using the mechanisms described in this paper.

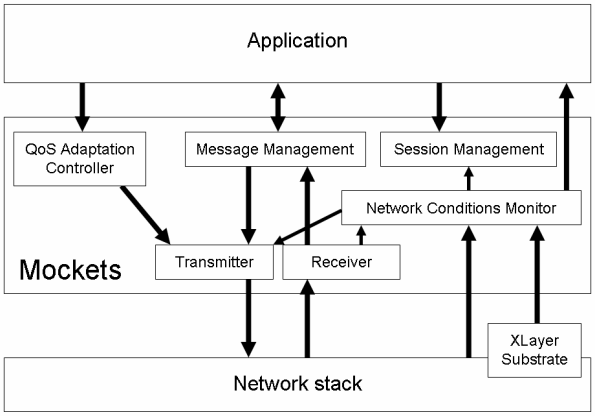

Figure 1. The Network Conditions Monitor component in the Mockets middleware.

V. MONITORING NETWORK LATENCY

Latency is an extremely important parameter in distributed applications. It is of paramount importance in time-sensitive, e.g., multimedia streaming or VoIP, and proximity-based applications, e.g., overlay construction or server redirection [11].

Network latency monitoring in Mockets is currently performed by measuring the RTT of established connections. Although the latency in TX and RX directions of a communication path can significantly vary, especially in MANET scenarios, assuming that network latency is half the RTT is an acceptable simplification.

Mockets implements three RTT measurement algorithms, with different trade-offs between their promptness of reaction to changes (agility), resistance to spurious fluctuations in the measure (stability) and resource consumption (overhead). None of the algorithms can be perfectly suited for every situation and working environment. Instead, applications can choose which algorithm to use according to their service logic, user preferences and current execution context, e.g., network status and battery level. Mockets was designed to be highly configurable and allows applications to choose the desired RTT estimation algorithm orthogonally. As of this writing, the choice must be done at compile time, but future versions of the middleware will permit switching between different algorithms at run time.

The RTT measurement implemented in Mockets are: *ACK-based*, *timestamp-based*, and *timestamp-based with compensation*.

The ACK-based algorithm simply measures RTT as the difference seen at the sender between the time an ACK is received and the time the packet was sent. Retransmitted packets are not considered in this measure.

The timestamp-based algorithm introduces a timestamp in every transmitted message. Receivers return the timestamp in the ACK. RTT is then calculated by the sender as the difference between the time an ACK is received and the timestamp. This algorithm considers retransmitted messages in RTT measurement and therefore is more agile than the previous one in the case of high packet loss rate.

The timestamp-based algorithm with compensation is a variation of the timestamp-based algorithm in which the receiver considers the message processing time in ACK responses and decreases the timestamp accordingly before returning the ACK to the sender. This algorithm is the most accurate in the case of heavy load or scarcity of computational resources at the receiver.

Since the RTT values obtained from measurement samples can exhibit abrupt and significant changes, Mockets tries to mitigate the influence of transient variations and

glitches by considering a smoothed RTT value. In particular, Mockets currently adopts the traditional RTT smoothing algorithm of TCP, which is based on an exponentially weighted moving average. This algorithm has good stability but poor agility characteristics and might not be best suited for MANET environments. We are evaluating other smoothing algorithms with different trade-offs between agility and stability.

VI. EXPERIMENTAL RESULTS

We have tested the performance of the 3 RTT measurement algorithms in order to point out their different behaviors in terms of agility and stability. We have performed all the experiments in an emulation environment where we can better control both variations in network parameters to estimate and the network conditions of the environment (e.g., packet loss rate).

The experimental testbed is composed of two Centrino 2.0 GHz laptops running Linux interconnected via a 100 Mbps 802.3 wired network. In the first test, one of the computers runs Netem, a network emulator included in the Linux kernel which can apply several effects to the outgoing traffic flow in order to reproduce the behavior/dynamics of a real wireless network.

We have configured Netem to vary values of network delay between the two hosts and to enforce a high packet loss rate (40%), in order to emulate a networking environment with highly unreliable communications, typical of the MANET scenario.

The first test compares the RTT measurement algorithms in the case of a single abrupt change in network latency. In particular, we have instantaneously increased the network delay parameter in the Netem-based emulation environment from 20ms to 40ms. Figures 2 and 3 plot the Smoothed RTT (SRTT) estimation performed by the Mockets framework against the delay imposed by Netem using the ACK-based and timestamp-based algorithms respectively.

There is a small but significant difference between the results obtained through the two algorithms. In fact, since it considers retransmitted packets in RTT measurement, the timestamp-based algorithm can capture higher dynamics in changes to network delay and is therefore more agile than the ACK-based one. Unfortunately, the RTT smoothing algorithm currently adopted in Mockets is not capable to take full advantage of the greater amount of information provided by timestamp-based RTT measurement algorithm.

We do not show the results obtained using the timestamp-based algorithm with compensation, as they are not significantly different from the ones presented in Figure 3. This is justified by the fact that in the testing environment there is no scarcity of computational resources at the receiver and therefore the two timestamp-based algorithms have the same behavior and expected outcome.

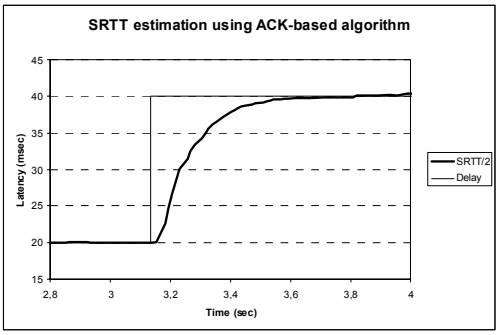

Figure 2. SRTT estimation using ACK-based algorithm in the case of a single abrupt change in network latency.

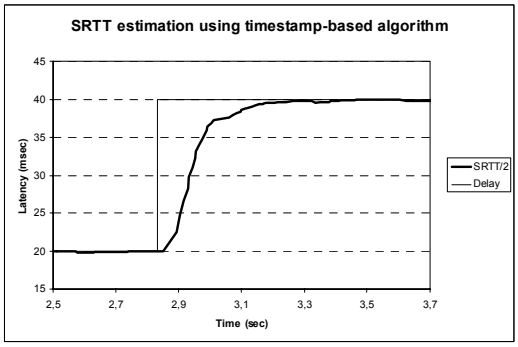

Figure 3. SRTT estimation using timestamp-based algorithm in the case of a single abrupt change in network latency.

In the second test, the laptops are connected via a Pentium 4 2.0 GHz PC running NISTNet, a network emulator which allows the enforcement of the same network effects supported by Netem, but with a finer-grained resolution. All the traffic between the laptops is routed through the NISTNet machine, which applies the desired effects to the incoming traffic flow. NISTNet was configured to enforce the same emulation parameters adopted for the previous test (40% packet loss).

The second test performs a comparison of the RTT measurement algorithms in the case of small but continuous variations in network latency. We have configured NISTNet to increase the network latency from 20ms to 40ms and then decrease it again to 20ms in steps of 1ms. Figures 4 and 5 plot the SRTT estimation performed by the Mockets framework against the delay imposed by NISTNet using the ACK-based and timestamp-based with compensation algorithms respectively.

The timestamp-based with compensation algorithm provides a more agile and accurate estimation of network latency than the ACK-based one. The difference between the two algorithms is more significant than in the previous experiment.

We do not show the results obtained using the timestamp-based algorithm as they are not significantly different from the ones obtained using the timestamp-based algorithm with compensation. As in the previous test, this is an expected result due to the abundance of computational resources at both end hosts.

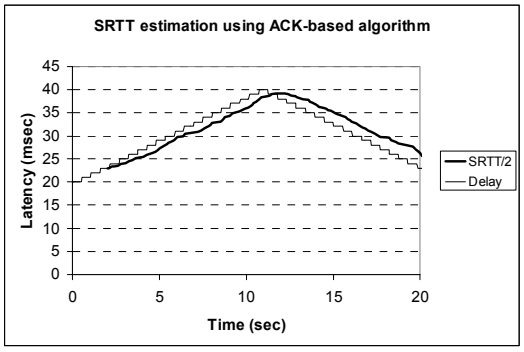

Figure 4. SRTT estimation using ACK-based algorithm in the case of slow continuous changes in network latency.

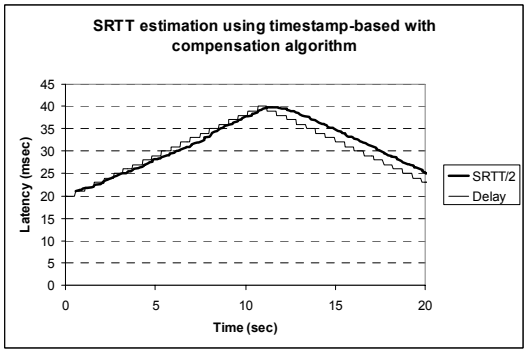

Figure 5. SRTT estimation using timestamp-based with compensation algorithm in the case of slow continuous changes in network latency.

VII. RELATED WORK

The design and implementation of the Mockets framework has required investigation in several areas of research. Network-awareness in dynamic environments is not a new concept in distributed systems research. In their influential work, Bolliger and Gross proposed a framework based on a feedback closed-loop that controls adjustment of an application to network properties [1]. However, their framework is only theoretical and they do not provide an actual implementation.

Many other research proposals have studied network conditions monitoring in dynamic distributed environments. However, most of these studies, such as the Remos project, perform centralized monitoring of resource availability/utilization [12] and are therefore not well suited for the MANET environment. Some other research projects, like Enable [13], simply focus on automatic fine tuning of TCP connections to improve applications performance without increasing their robustness and resilience to variations in network conditions. Other proposals perform both monitoring and QoS adaptation. However, most of theses studies such as Odyssey [14] and A^3 [15] require the deployment of a dedicated QoS adaptation component for every type service, thus preventing the realization of common adaptation strategies for different services. Mockets goes beyond these limitations and provides a network-aware programming model which enables the development of robust and adaptive applications on MANETs by means of decentralized end-to-end network conditions monitoring and service-agnostic QoS adaptation mechanisms.

With regards to network latency estimation, research proposals can be divided in two categories. Proposals of the first category provide one-way latency between two hosts by calculating their distance in a coordinate space where Euclidean distance represents latency in communications [11], [16]. This approach requires the deployment of reference servers with a fixed and well-known position in the coordinate space and is therefore not well suited for the MANET scenario.

The second category of proposals calculate network latency via RTT estimation. In particular, researchers have especially focused on the realization of RTT smoothing algorithms with good stability and agility characteristics [17], [18]. Mockets adopts the same approach and, in addition, integrates the network latency estimation in a framework to support the realization of QoS-adaptive applications in MANETs.

VIII. CONCLUSIONS AND FUTURE WORK

The Mockets middleware provides applications with a network-aware programming model suitable to the highly dynamic MANET environment. The Mockets Network Conditions Monitoring presented in this paper naturally complements the QoS adaptation mechanisms, thus enabling Mockets applications to perform informed service tailoring decisions in the case of abrupt changes in the quality of the communication channels.

We are currently working on the Mockets network monitoring component in order to introduce new latency

estimation algorithms (i.e., the latency estimation for single-direction of communication) and to extend the network parameters measurements including the available bandwidth and network capacity.

We are also realizing a module for the ns2 simulator implementing Mockets-based communications, so that we can run performance comparisons of our work with other related proposals under the same working conditions.

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, by the Air Force Research Laboratory under Cooperative Agreement FA8750-06-2-0064, and by the Italian MIUR in the framework of the Project "MOMA: a middleware approach to MObile MultimodAl web services".

REFERENCES

[1] J. Bolliger, T. Gross, A Framework-Based Approach to the Development of Network-Aware Applications, *IEEE Transactions on Software Engineering*, Vol. 24, No. 5, pp. 376-390, May 1998.

[2] M. Sarela, P. Nikander, Applying Host Identity Protocotl to Tactical Networks, in *Proceedings of 23rd Military Communications Conference (MILCOM 2004)*, Atlantic City, NJ, USA, October 2004.

[3] J. Cao, K. M. McNeill, D. Zhang, J. F. Jr. Numaker, An Overview of network-aware applications for mobile multimedia delivery, in *Proceedings of 37th Annual Hawaii International Conference on System Sciences*, 2004.

[4] N. Suri, M. Tortonesi, M. Arguedas, M. Breedy, M. Carvalho, R. Winkler, Mockets: A Comprehensive Application-Level Communications Library, in *Proceedings of 24th Military Communications Conference (MILCOM 2005)*, Atlantic City, NJ, USA, October 2005.

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

[6] N. Suri, J. Bradshaw, M. Carvalho, T. Cowin, M. Breedy, P. Groth, R. Saavedra, Agile computing: bridging the gap between grid computing and ad-hoc peer-to-peer resource sharing, in *Proceedings of 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2003)*, pp. 618-625, 12-15 May 2003.

[7] N. Suri, M. Carvalho, J. Lott, M. Tortonesi, J. Bradshaw, M. Arguedas, M. Breedy, Policy-based Bandwidth Management for Tactical Networks with the Agile Computing Middleware, in *Proceedings of 25th Military Communications Conference (MILCOM 2006)*, Atlantic City, NJ, USA, October 2006.

[8] B. Landfeldt, P. Sookavatana, A. Seneviratne, The Case for a Hybrid Passive/Active Network Monitoring Scheme in the Wireless Internet, in *Proceedings of the 8th IEEE International Conference on Networks (ICON 2000)*, pp. 139-144, 2000.

[9] M. Carvalho, N. Suri, V. Shurbanov, E. Lloyd, A Cross-layer Network Substrate for the Battlefield, in *Proceedings of the 25th Army Science Conference*, Orlando, FL, USA, 2006.

[10] R. S. Prasad, M. Murray, C. Dovrolis, K. Claffy, Bandwidth estimation: Metrics, measurement techniques, and tools, *IEEE Network*, 2003.

[11] S. Ratnasamy, M. Handley, R. Karp, S. Shenker, Topologically aware overlay construction and server selection, in *Proceedings of IEEE INFOCOM'02*, New York, NY, USA, June 2002.

[12] B. Lowekamp, N. Miller, R. Karrer, T. Gross, P. Steenkiste, Design Implementation and Evaluation of the Remos Network Monitoring System, *Journal of Grid Computing*, Vol. 1, No. 1, , pp. 75-93, May 2003, Kluwer.

[13] B. L. Tierney, D. Gunter, J. Lee, M. Stoufer, J. B. Evans, Enabling Network-Aware Applications, in *Proceedings of 10th IEEE International Symposium on High Performance Distributed Computing (HPDC-10 '01)*, San Francisco, CA, USA, 2001.

[14] B. Noble, System Support for Mobile, Adaptive Applications, *IEEE Personal Communications*, Vol. 7, No. 1, February 2000.

[15] Z. Zhuang, T. Chang, R. Sivakumar, A. Velayutham, A^3: Application-Aware Acceleration for Wireless Data Networks, in *Proceedings of the 12th annual international conference on Mobile computing and networking (MobiCom 2006)*, Los Angeles, CA, USA, 2006.

[16] K. Gummadi, S. Saroiu, S. Gribble, King: estimating Latency between Arbitrary Internet End Hosts, in *Proceedings of the SIGCOMM Internet Measurement Workshop (IMW 2002)*.

[17] M. Kim, B. Noble, Mobile Network Estimation, in *Proceedings of 7th Annual Conference on Mobile Computing and Networking*, July 2001.

[18] K. Jacobsson, H. Hjalmarsson, Niels Möller, K. H. Johansson, Round-Trip time estimation in communication networks using adaptive Kalman filtering, in: *Reglermöte 2004*, Gothenburg, Sweden, May 2004.