Network Protocols for Mobile Robot Systems

Douglas W. Gage

Space and Naval Warfare Systems Center San Diego
SPAWARSYSCEN D371, San Diego, CA 92152-7383

ABSTRACT

Communications and communications protocols will play an important role in mobile robot systems able to address real world applications. A poorly integrated "stack" of communications protocols, or protocols which are poorly matched to the functional and performance characteristics of the underlying physical communications links, can greatly reduce the effectiveness of an otherwise well implemented robotic or networked sensor system. The proliferation of Internet-like networks in military as well as civilian domains has motivated research to address some of the performance limitations TCP suffers when using RF and other media with long bandwidth-delay, dynamic connectivity, and error-prone links. Beyond these performance issues, however, TCP is poorly matched to the requirements of mobile robot and other quasi-autonomous systems: it is oriented to providing a continuous data stream, rather than discrete messages, and the canonical "socket" interface conceals short losses of communications connectivity, but simply gives up and forces the Application layer software to deal with longer losses. For the Multipurpose Security and Surveillance Mission Platform (MSSMP) project, a software applique is being developed that will run on top of User Datagram Protocol (UDP) to provide a reliable message-based Transport service. In addition, a Session layer protocol is planned to support the effective transfer of control of multiple platforms among multiple control stations.

Keywords: mobile robots, security sensors, surveillance, Internet, TCP/IP, UDP, reliable transport, datagram

1. INTRODUCTION

1.1 Communications in mobile robot systems

Communications will play an increasingly important role as mobile robots begin to be packaged into systems that can satisfy the requirements of real world applications. A mobile robot system might employ communications in any or all of the following modes:

- Between a robot and a user "control display" station, to send command tasking from the user (and the user's software proxies) to the robot and to receive back from the robot system status and environmental data, as well as data relevant to the robot's mission. In a security or surveillance system, for example, this mission data would include brief alarms and alerts (resulting from onboard processing of sensor data) indicating the detection of a possible threat, and additional data to permit assessment of the threat condition, such as sensor data "snapshots" (images and audio clips) and "streams" (video or audio).

- Between robots in a multi-robot system -- to support local sensor data fusion, to provide cueing from one sensor to another, to support both planning and coordinated execution of cooperative behaviors between robots, and/or to allow one robot to serve as a communications relay for other robots and/or user stations.

- Between the robotic system and external clients, by supporting the transfer of sensor data or other processed information from the robot system to, for example, superior military commanders. This data transfer could be handled in a number of ways: from the robots directly, from the user control display station, or via some sort of gateway -- this latter method could be used, for example, to make the data available to any authorized user with a standard Web browser such as Netscape.

- Between robots and the robotic system developer, to reduce technical risk and increase implementation productivity by providing software downloading and system debugging tools to exercise and validate hardware as

well as software. It should be possible for subsystem developers to design, implement, and integrate their software with actual robot hardware and other software modules, all from their own offices via the Internet.

While the work described in this paper has been performed in the context of military tactical security and surveillance sensor systems, involving both indoor and outdoor unmanned ground vehicles, airborne vehicles, and man-portable sensor packages, the results should be fully applicable to other mobile robots and quasi-autonomous (continuously sensing and behaving) systems addressing other application domains.

1.2 An example of protocol failure in a mobile robot system

The Mobile Detection Assessment and Response System (MDARS) is a joint Army-Navy development effort to provide an automated intrusion detection and inventory assessment capability for use in DoD warehouses and storage sites. The MDARS goal is to provide multiple mobile robotic platforms that perform random patrols within assigned areas in order to: (1) detect anomalous conditions such as flooding or fires; (2) detect intruders; and (3) determine the status of inventoried items through the use of specialized RF transponder tags. Separate Interior and Exterior development efforts address the requirements of DoD warehouses and outdoor DoD storage sites, respectively.

Communications plays an important role in the MDARS system because (1) MDARS must function as a key component of a complete security system that also includes fixed detection capabilities and human security guards; and (2) the patrol coverage of a large number of mobile robotic platforms must be controlled and coordinated to minimize opportunities for undetected intrusion, even by insiders. This requirement to support up to 32 platforms at the same time motivated adoption of a navigation approach based on centrally planned routine patrol routes, with any deviations handled on an exception basis. This feature is implemented in MDARS via the Multiple Resource Host Architecture (MRHA), in which a LAN interconnects a Supervisor computer, a pool of centrally located Planner/Dispatcher computers, one or more (human guard) Operator Stations, and a Link Server to route messages to and from the robots via RF. When a robot becomes idle, the Supervisor assigns a Planner/Dispatcher from the pool to plan and download a new path. The robot then autonomously executes the set of commands until completion or until an exception condition is encountered, whereupon the process repeats. Meanwhile, the Planner/Dispatcher is released back to the pool for allocation to another platform, so that a small number of Planner/Dispatchers can handle a large number of robots. When an event is deemed to require the attention of a human operator, the Supervisor assigns an Operator Station, which displays the appropriate information to the guard and provides an interface for command entry. Details of the MDARS system and the MRHA architecture can be found in papers[1,2,3] by Everett et al.

In 1993, the MDARS Interior program began implementation of a testbed installation in a large warehouse at Camp Elliott, near San Diego CA. COTS ("Commercial Off The Shelf") Arlan RF modems were installed to provide a 9600 bps serial data channel to each robot, but this serial link was implemented via an underlying scheme of data packetization and automatic error detection and data retransmission which was transparent to the user processes.

It was observed that the RF communications link between the robot and the control station's Link Server became very unreliable whenever the robot moved into any of several specific areas, and a subsequent systematic mapping of RF signal levels revealed wide variation throughout the warehouse, including areas of very low signal strength. Whenever the robot entered one of these RF nulls, the Arlan RF units would automatically continue to retransmit packets until the data got through. This retransmission was transparent to the higher level processing written by SPAWARSYSCEN (then NRaD), except that the higher level protocol had its own one-second timeout which would occasionally be exceeded, resulting in the higher level's retransmission of the complete message. Because the character-oriented data stream was protected by only a simple 8-bit checksum (inherited from the COTS Cybermotion *K2A Navmaster* mobility platform), the higher level processing would occasionally accept improperly combined message fragments as a valid message, resulting in the trashing of various status bits (incorrectly indicating a low battery condition, presence of an intruder, a fire alarm, etc.). In one case, a "minor" communications error led the system to initiate a sequential interrogation of 2000 nonexistent inventory tags.

This problem, basically caused by an unforeseen interaction between essentially incompatible COTS system hardware, software, and firmware elements, was addressed by moving to a much higher bandwidth Arlan Model 640 RF Ethernet implementation, adopting the use of TCP/IP protocols, and incorporating a more robust CRC in the higher level protocol. Just as for the MSSMP case to be discussed below, MDARS is now moving from TCP/IP to UDP/IP (with the application

software providing message acknowledgement and retransmission services) in order to reduce software complexity caused by TCP's stream orientation and the transparency of its error recovery behavior.

1.3 IP-based RF networks for mobile robot systems and military applications

COTS wireless Ethernet transceiver and bridge technologies such as the Arlan devices adopted by the MDARS project provide a number of features which are very attractive for mobile robot systems, including self-organizing networking, data packet autorouting, mobile roaming/handoff, and remote SNMP configuration control. Automatic network configuration capability enables idiot-proof network connectivity even in a highly dynamic mobile environment. The tree structured Spanning Tree Protocol (IEEE 802.1d) is an example of an open standard self-organizing network protocol used in COTS IP modems. Each radio maintains a dynamic configuration table of who is directly connected to whom within the wireless network. Radios are automatically added to or deleted from the network configuration table as they enter or leave the network. IP devices connected to each radio are also listed in the configuration table. This allows dynamic point-to-point paths between devices across the wireless network to be automatically established by the radios without the need of operator intervention. Beyond-line-of-sight endpoints can maintain connectivity across this dynamic network through data packet multihopping, autorouting, and autohandoff (roaming) features. If a radio node goes down, radio configuration tables will be updated and the data packets which would have been passed through the down radio node will be rerouted automatically.[4]

A number of military programs, including DARPA's Warfighter Internet program, the Army's Wireless Networking Initiative, and ONR's Wireless Communications 6.2 Program, are investigating how to adapt this COTS wireless technology for use by the military. These military programs also focus on developing technologies to augment COTS wireless capabilities in the areas of security, military band frequency transition, and distributable master routing tables.

On the grand scale, the DoD concept for joint services interoperability in the 21st Century, *C4I For The Warrior*[5], envisions a widely distributed user-driven infrastructure in which the warrior "plugs in" to obtain information from secure and seamlessly integrated Command Control Computer Communications and Intelligence (C4I) systems. Each Service has its own strategy for meeting this vision, and IP protocol compliance has been designated as the glue between all of these strategies to obtain and maintain interoperability between the services, and to provide the "plug in" capability to the warrior. The *Army Digitization Master Plan (ADMP)*, the Army's roadmap for fulfilling its *Enterprise* strategy[6], calls for the establishment of a seamless communication infrastructure called the "Tactical Internet" as an integrated battlefield communications network which is focused primarily on achieving seamless information transfer both vertically and horizontally across the battlespace for Brigade and Below Units. The ADMP further states: "The common thread in Army digital data communications will be the use of Internet protocols. Transmission Control Protocol/User Datagram Protocol (TCP/UDP) and Internet Protocol (IP) are used at layers 4 and 3, respectively."

2. INTERNET PROTOCOLS: LAYERING, CHARACTERISTICS, DEFICIENCIES

2.1 Protocols and Protocol Layers

A communications protocol is essentially a language used by two entities to exchange information over a communications channel -- it represents a shared understanding or agreement of how each entity will interpret the signals it receives from the other. Even the use of a simple standard asynchronous RS-232 link requires agreements as to (1) which conductors of the cable (i.e., which connector pins) will carry the data stream in which direction, and which, if any, conductors will carry control information (and how the receiver of control information should respond to it), (2) the voltage levels used to represent a 1 and a 0, (3) the link signalling rate, or time a given voltage will be maintained in order to represent a single bit of information, and (4) the representation of a character: the presence of a start bit, the minimum number of stop bits, whether or not there is a parity bit (and how to compute it), and whether the data bits are ordered with the most- or the least- significant bit first. Based on all these agreements, a stream of characters transmitted by a device connected to one end of the cable is received and interpreted as the same stream of characters by another device connected to the other end.

Of course, if the communications between our two serially-interfaced devices is to be useful, additional agreements must exist as to the "meaning" of the data characters being exchanged, and these agreements -- as to which characters or strings of characters should represent different commands or responses to commands -- should be independent of the *physical layer*

agreements defining the physical representation of the characters. This is the key notion of protocol layering: that the complete set of agreements between two communicating systems can be divided into a number of independent hierarchical layers, with "protocol entities" at each layer using a service provided by the interaction of the protocol entities at the next lower layer. See Ross[7] for a good concise discussion of protocol layering.

The Open Systems Interconnection (OSI) Reference Model[8], developed by the International Standards Organization (ISO) in the early 1980s, was intended to be general enough to serve as the basis for description of even the most complicated network systems. This model defines seven layers (from lowest to highest): Physical, Link, Network, Transport, Session, Presentation, and Application. A number of specific protocols have been developed as implementations of the ISO OSI Reference Model, but the explosive growth of the Internet has focused the attention of developers on Internet-compatible protocols.

2.2 The Internet Protocol "Stack"

In late 1969, two computers in Southern California were interconnected through leased phone lines as the first step in a packet switching experiment sponsored by the Advanced Research Projects Agency (ARPA). The experiment was a success, and by the mid-1970s dozens of nodes in universities and government and commercial laboratories were connected to the ARPANET, allowing users to exchange email, transfer files, and remotely log in to distant host computers. As its exponential growth continued, and ARPA's involvement in the project ended, the network of thousands of nodes became known as the Internet. The proliferation of workstations and personal computers drove the evolution of network-based applications from a terminal-host model to a client-server one, culminating in the early '90s appearance of an application that combined hypertext-based multimedia with free client software that ran on almost any workstation or PC; in 1995 this "killer app", the World Wide Web, exploded the multi-million node Internet into the popular consciousness and culture.

The protocol "stack" that supports the Internet consists of the following layers, from the lowest to the highest:

> **Physical Layer:** Defines the mechanism, such as an Ethernet cable, a telephone connection between two modems, or a simple wire (described in Section 2.1 above), that transports a physical representation of the stream of digital bits from one node to another.
>
> **Link Layer**, or **Data Link layer**: Transports packets of data bits across a simple link or a single network hop. Usually involves some sort of checksum for error detection, and, for shared media such as a data bus, performs Media Access Control (MAC) functions such as Ethernet's CSMA/CD (IEEE 802.3) or token passing on a ring (IEEE 802.4) or bus (IEEE 802.5). Serial Line IP (SLIP) and Point to Point Protocol (PPP) are Link layer protocols commonly used on dialup phone lines. High-level Data Link Control (HDLC) is often used over synchronous serial links.
>
> **Network Layer** and **Transport Layer**: discussed below.
>
> **Application Layer**: The workstation or PC user actually interacts with application programs which make use of specific Application protocols: HTTP (World Wide Web), SMTP and POP (email), FTP (file transfer protocol), Telnet (remote login), and NNTP (network news) are the classics among the expanding number of Internet Application layer protocols.

The principal difference between the Internet and OSI protocol stacks is that the functions of the OSI Presentation and Session layers, to the degree that they implemented either explicitly or implicitly, are subsumed into the Application layer in the Internet protocol model.

One key factor that allowed the Internet to grow rapidly, steadily, and nondisruptively over many orders of magnitude was the upgrading of its core protocols in the early 1980s, with a clear demarcation between the Network layer's Internet Protocol (IP)[9] that makes a "best effort" to route packets from node to node through the Internet to the intended destination, and the Transport layer's Transmission Control Protocol (TCP)[10], that keeps track of the packets in each end-to-end connection.

The Network and Transport Layer protocols are designed to provide complete isolation between the Application layer above and the Link and Physical layers below. File Transfer Protocol (FTP)[11], for example, is an application that moves a file from one internetted node to another, without having to know whether the underlying network connectivity is via direct Ethernet at 10 Mbps, via leased T1 line at 1.5433 Mbps, or via dial up modem at 28.8 kbps. TCP, in particular, works very hard to maintain this isolation: it buffers data and resends lost packets and discards duplicate packets and manages the flow of packets carefully to avoid overflowing buffer capacity or overloading the network, while completely hiding all this complexity from the user. This process generally works very well in the commercial Internet environment, and when it doesn't work well, all the user sees is delay in the delivery of the data -- the complexity still remains hidden. (Performance failures are most often associated with the lowest bandwidth and least controlled elements of the overall system: e.g., the dialup phone connection to one's Internet Service Provider.)

The problem for military communications users is that even $100 commercial 28.8 kbps telephone modems look like Star Wars technology when compared to current military tactical RF communications capabilities. The use of low bandwidth high error-rate links in an environment in which RF link interconnections between mobile nodes dynamically come and go, and in which an adversary may be trying very hard to deliberately destroy our communications assets, creates an IP network environment very different from that of the commercial Internet.

This work addresses some of the issues associated with using limited performance IP networks to support tactical robotic and security/surveillance sensor applications -- in other words, it seeks to determine the best ways for tactical mobile robots and sensor nodes to use IP-based networks to get the biggest "bang for the bit". Do the protocols that have been developed for the commercial Internet make sense in the tactical environment? What are the requirements for new protocols?

2.3 Existing Internet Transport Layer Protocols

The Network layer's IP[9] protocol provides the mechanism by which a packet transmitted by a source node makes its way through potentially many transfers from node to node until it is received by the intended destination node. The IP layer processing within each node doesn't keep track of whether a packet it is sending actually ever gets to its destination -- it just passes each packet it receives on to whichever of its neighbors it thinks is "closest" to the final destination address. User processes access this IP layer mechanism through any of a number of Transport layer protocols, which provide services of widely varying levels of capability. The two most important Transport protocols, however, are TCP and UDP.

Transmission Control Protocol (TCP)

TCP is the workhorse Transport protocol of the Internet, so much so that the Internet protocol suite is usually referred to as "TCP/IP". The TCP service model is that of a virtual connection -- a user process passes a stream of data to TCP on the source node, and TCP promises to deliver the exact same stream to the specified destination user process, wherever it may be. This means that TCP must decide how to divide the input user data stream into packets, store and keep track of each packet it sends, retransmit packets which remain unacknowledged for a timeout period, acknowledge received packets (while discarding duplicates), and reassemble received packets into a data stream for the user process, all the while making it appear to the user processes that they have a simple serial connection between them. This is a complex business, and the TCP specification, RFC 793[10], is 85 pages long. Nevertheless, a TCP/IP package runs on every networked workstation or modem-equipped PC that supports browser access to the World Wide Web or a desktop email program like Eudora.

User Datagram Protocol (UDP)

UDP is a Transport layer protocol that is much simpler than TCP and that is used to support many "behind the scenes" functions on the Internet, such as Domain Name Server (DNS). UDP allows user processes almost direct access to basic IP functionality; i.e., to send and receive packets to processes on other Internet nodes. UDP packets specify source and destination "ports", so that packets can be delivered to the appropriate process on the destination node, and include a checksum, so that the receiving process can be sure that the packet it has received is correct. Beyond this, it's just raw IP packet delivery. The UDP specification, RFC 768[12], is less than 3 pages long. UDP is supported as part of all standard TCP/IP software packages.

In addition to TCP and UDP, a number of Transport layer protocols have been or are being developed to support a variety of specialized applications, such as RTP (Real Time Protocol[13]) and RMTP (Reliable Multicast Transport Protocol[14]). One of these which is relevant to the requirements of the mobile robot application is Reliable Data Protocol (RDP).

Reliable Data Protocol (RDP)

Developed by BBN for DARPA in the middle 1980s, RDP is a Transport layer protocol which was designed to efficiently support the bulk transfer of data for such host monitoring and control applications as loading/dumping and remote debugging. RDP defines a service model more appropriate to mobile robot applications (and some other tactical applications) than TCP's: the reliable delivery of message (not byte-stream) traffic. RDP also makes more efficient use of limited communications bandwidth than TCP, and is much simpler to implement.

Unlike TCP and UDP, however, which are fully adopted Internet Standard Protocols whose implementation is "recommended" (i.e., necessary if a host node is to support user processes) and whose specifications are very stable (RFC 793 for TCP was issued in September 1981, RFC 768 for UDP in August 1980), RDP is an "Experimental Protocol" whose implementation status is "Limited Use". RDP was specified in RFC 908[15], dated July 1984, supplemented (rather than replaced) by RFC 1151[16] dated April 1990.

One wonders why a reliable message-oriented protocol like RDP isn't used across the wide range of Internet applications -- including email, World Wide Web, and FTP -- that require the transport of messages or files, but which in fact utilize a byte-stream oriented TCP connection. The answer is that Telnet (remote login) was widely used on the ARPANET of the middle 1970s, and TCP was developed to support Telnet's requirement for a reliable stream transport with response time short enough to satisfy the remote keyboard display user. Developers of the message-oriented Application layer services that emerged later found it easier to simply use the reliable Transport layer mechanism that was already available than to develop something new.

2.4 Limitations of TCP

2.4.1 TCP Performance Deficiences

While the operative RFC for TCP dates from 1981, and TCP has worked well enough to support the phenomenal growth of the Internet, a number of important performance deficiencies have been identified in TCP, and corresponding "fixes" have at least been designed and in some cases implemented as options to the basic protocol.

While TCP's basic goal is to convey data to its destination as expeditiously as possible, it is supposed to do this with minimum impact on the global performance of the Internet as a whole. "Reliable Transport" means that TCP must store all its outgoing data until its receipt at the destination is acknowledged, so that it can retransmit packets that are lost. The problem is that a packet can be "lost" in either of two ways: first, an error in some communications channel may cause the packet to be discarded along its way by the IP layer, or second, it may be discarded by some node because that node doesn't have buffer space to store it. If a packet is lost due to a communications link error, it should be retransmitted as soon as possible; if, on the other hand, the loss is due to congestion in some portion of the network, then it may be important to delay retransmission to avoid exacerbating the problem.

In order to support reasonable throughput over a TCP connection, TCP implements a sliding window scheme whereby each receiving node tells its partner how many bytes it is allowed to send before it receives an ACK. As the receipt of packets is acknowledged, the sender "slides" the window by the number of bytes acknowledged. If the window size is larger than the channel rate times the round trip delay time, then the sender can continuously send data and maximum throughput is achieved. For high bandwidth, long time delay connections via geosynchronous satellite, this can be a lot of data, all of which must of course be buffered until ACKed. If we want to use TCP/IP to communicate with distant spacecraft[17], the problem gets even nastier, since, if multiple packets are lost, then the sender has to wait for an extremely long timeout before retransmitting all the packets starting with the first unacknowledged one.

The solution to this last problem is to use a *selective* acknowledgement (SACK) so that if a node receives a packet whose sequence number indicates that one or more previous packets have indeed been lost, it can identify the lost packet(s) for retransmission while ACKing the data that it did receive. SACK is an organic feature of RDP, and SACK Options have been discussed for TCP for many years. The most recent formal proposal for TCP SACK is RFC 2018, dated October 1996.

A number of other important mechanisms to improve TCP connection throughput and to minimize network congestion have been proposed over the past years: scaled windows (to prevent sequence number wraparound), timestamps (for explicit round trip time measurement), Fast Retransmit, Fast Recovery, and so on. See RFC 1323 and RFC 2001.

2.4.2 Inappropriate TCP Service Model

Beyond intrinsic performance issues, TCP has some other disadvantages for applications that only need to exchange messages. The specification for RDP, RFC 908, states:

> - TCP is oriented toward a more general environment, supporting the transfer of a stream of bytes between two communicating parties. TCP is best suited to an environment where there is no obvious demarcation of data in a communications exchange. Much of the difficulty in developing a TCP implementation stems from the complexity of supporting this general byte-stream transfer, and thus a significant amount of complexity can be avoided by using another protocol. This is not just an implementation consideration, but also one of efficiency.
>
> - TCP provides sequenced delivery of data to the application. If the application does not require such sequenced delivery, a large amount of resources are wasted in providing it. For example, buffers may be tied up buffering data until a segment with an earlier sequence number arrives. The protocol should not force its segment-sequencing desires on the application.

RFC 908 then observes:

> RDP supports a much simpler set of functions than TCP. The flow control, buffering, and connection management schemes of RDP are considerably simpler and less complex. The goal is a protocol that can be easily and efficiently implemented and that will serve a range of applications.

In general, these considerations also characterize the Transport requirements of the mobile robot application.

2.5 Interfaces between protocol layers: the socket interface

A protocol defines the interaction between peer-entities in a specific protocol layer, which, making use of services provided by the next lower layer, in turn provides a service to the next higher layer[7]. For example, TCP peer entities at the Transport layer use IP services to provide TCP service to POP3, HTTP, FTP, and a whole set of other Application layer users, while UDP peer entities make use of the same underlying IP services to provide UDP -- a different Transport layer service -- to TFTP and the rest of a different set of Application layer users.

The details of *how* the protocol entities at one layer actually access the service provided by the next lower layer is defined as the "interface" between the layers, *not* a protocol specified in an Internet RFC. To the Application level programmer using a procedural language such as C, the Transport layer interface provided by a commercial TCP/IP package consists of a number of functions which can be called to open and close connections and send and receive data. Needed information is passed as function arguments, and once a buffer of data is passed to the TCP/IP package for transmission, the communications process for that buffer is completely invisible to the Application level process that called the "net_write" or equivalent function. The ad-hoc standard is the "socket" interface that first appeared in BSD (Berkeley Software Distribution) 4.1c Unix in 1982, and has now spread well beyond Unix -- hence the "WinSock" used in Windows-based PCs[18,19]. The socket interface allows the application programmer to send and receive I/O with the same basic techniques s/he uses to write and read files.

The normal concern in defining a layer interface is to provide transparency and hide the details of the lower layer's processing from the higher layers -- but we will argue below that the higher levels of control of a mobile robot or other continuously

behaving system *need* to know some of what's actually happening in the communications processes supporting it. Since we are using IP as a given common Network layer, then it will be the responsibility of our Transport layer mechanism to ensure that Application layer processes can learn what they need to know about what is happening in the communications subsystem.

3. PROTOCOLS FOR THE MSSMP SYSTEM

3.1 The Multipurpose Security and Surveillance Mission Platform (MSSMP)

The Multipurpose Security and Surveillance Mission Platform (MSSMP)[20], initiated in 1992 as the Air-Mobile Ground Security and Surveillance System (AMGSSS), is designed to provide a rapidly deployable, extended-range surveillance capability for a variety of operations and missions, including: fire control, force protection, tactical security, support to counterdrug and border patrol operations, signal/communications relays, detection and assessment of barriers (i.e., mine fields, tank traps), remote assessment of suspected contaminated areas (i.e., chemical, biological, and nuclear), and even resupply of small quantities of critical items. An MSSMP system consists of three air-mobile remote sensing packages and a base station.

The current design of the air-mobile platforms is based on the Sikorsky Cypher ducted-fan vertical-take-off-and-landing unmanned air vehicle. This air-mobile platform carries its sensor package from one ground surveillance location to another, up to 10 km from the base station. MSSMP system requirements include high mobility, remote operations over low-bandwith tactical radio links, long-endurance surveillance capabilities, and the ability for one operator to supervise several remote systems. MSSMP sensor packages may also be deployed as man-portable stand-alone units, as well as being mounted on air-mobile platforms.

Each sensor package is mounted on a pan-and-tilt unit, and includes a visible light video camera, infrared video camera, and laser range finder. In addition, a serial port is provided to interface to an optional portable acoustic sensor. To minimize radio traffic, most sensor processing is performed by the remote payload. Acoustic and visual motion detection is used to detect, identify, and locate targets of interest. Preprogrammed responses are activated upon detection and may consist of only a simple alert to the operator, or may also include the automatic transfer of a static image, laser range or an image stream.

For the prototype unit, the operator's control display station is a laptop computer running a graphical Windows program. Commands to the remote sensors are initiated using the laptop's keyboard and pointing device, and returned data and images are displayed on the laptop's color display.

The communications approach initially explored for the MSSMP utilized military SINCGARS radios and prototype PCMCIA Tactical Communication Interface Modules (TCIMs) from Magnavox. (SINCGARS is a frequency hopping or single channel VHF-FM radio that operates in the 30 - 88 Mhz frequency range. It provides only 16kbps throughput, and takes several hundred msec to switch between transmit and receive modes.) Field tests conducted in 1995 showed that SINCGARS radios were not an effective means of communicating "high bandwidth" data like imagery/video. MSSMP therefore moved to COTS Arlan 640 Ethernet bridges as the basis for communications between all remote payload subsystems and the control/display station.

3.2 Goals for MSSMP Communications

MSSMP requires a communications architecture that can provide a full spectrum of required generic communications functionality, placing maximum reliance on COTS hardware and software and standard protocols, while requiring a minimum of additional mechanism. Communications performance goals for this project are aimed at providing enough throughput to accommodate mid-resolution imagery, reduced frame rate video, and processed alarms/alerts, while still maintaining a responsiveness that will be acceptable to a human user. Specific goals include:

- Transparent support for communications with both local and distant processors. We plan to use the same Transport layer communications mechanisms between processors located on the same physical platform (and hence on the same physical Ethernet segment) as well as with those on other platforms (remote Ethernet segments connected, in our MSSMP system, via Arlan 640 Ethernet Multipoint Bridges). This supports easy system

reconfigurability and provides a clean and simple communications mechanism to support system debugging and subsystem diagnostics in the field.

- Tailoring to the design communications topology. If a processor sends a message to another processor located on the same physical Ethernet segment and the message is not acknowledged immediately, then we want to retransmit the message as quickly as possible, and continue until it is received. If a message sent via low bandwidth RF to a process on another physical platform is not acknowledged, then we want to be more careful about how we use RF channel capacity in retransmitting the data (how soon/often, how many times to retransmit the message).

- Adapting to the current communications topology. If another platform doesn't acknowledge *any* of the messages we send to it over a period of time, then our system should be able to make the inference that either the other node or the channel is down. This capability may be implemented within the communications subsystem to a greater or lesser degree, but in any event communications performance data must inform the behavior of the overall system, which might appropriately switch into a more autonomous mode of operation (affecting the strategies for data collection, data processing, and data distribution).

- Prioritization of message transmission, including automatic handling of perishable data. A threat alarm/alert should have a higher priority than a routine status message, and a newer alarm may or may not be more important than an old one. The MSSMP Payload Processor (PP) sends the Control Display (CD) a status message every 5 seconds. If the link to the CD is lost for a minute, the PP should *not* waste communications bandwidth sending (in order!) twelve queued obsolete status messages. When a new status message is created, it should be possible to suppress the transmission of any previous status messages that have not yet been sent.

3.3 MSSMP Transport Protocol

The MSSMP Transport layer protocol (let's call it "MTP" for convenience) development draws from three distinct sources:

- The definition, design, and implementation of the RDP protocol, as an example of a reliable message-based Transport service including a system for buffering the messages to be transmitted, message-oriented (as opposed to TCP's byte-oriented) sequence numbers, retransmission timeouts, and selective acknowledgements.

- The various performance improvement schemes being pursued as TCP options (in RFC proposals, designs, and even an occasional implemention) by the community of Internet protocol "wizards", as mentioned in Section 2.4.1 above.

- Development of a Transport layer service interface to address the concerns and goals discussed in Section 2.5 and Section 3.2 above.

While the first two of these derive completely from the previous work of others, and will not be discussed further here, the third represents the novel thrust of our development effort.

The MTP interface provides the Application layer programmer with a generalized "socketlike" interface, supplemented with read-only access to data which would otherwise be completely internal to the implementation. In brief, the two modes of generalization are: (1) when a process "opens" a connection, the connection stays open regardless of the status of the communications channel, until the process actively closes it, and (2) a process that sends a message can monitor its status and affect the transmission process up until the time that an ACK is received from the destination process. The interface will be designed so that an application process which has been written to use a standard socket based TCP/IP package (either using UDP, or using TCP on a strictly message-oriented basis) can use MTP in a default mode with having to worry about the generalizations.

When a connection is opened, MTP sets up the necessary data structures, allocates buffer space, and sends a UDP message to a well known UDP port on the destination host, which then sets up the connection on that end. When the application process on the destination opens the reciprocal connection, it is identified as having been already remotely opened (i.e., no distinction is made between active and passive opens). If the destination host does not answer, then the connection is still

"open", but its communications status is marked as "down" (or, more precisely, "not yet up"), and MTP will periodically attempt to make contact. The open command may also include system configuration information about the network topology between the nodes, and nominal link performance characteristics. If at any point MTP finds that it is unable to communicate, then it marks the connection as down -- but while the application process can learn that the connection is down (and can, in fact, ask MTP to signal whenever a connection goes down or comes back up) it does not have to keep track of trying to reinitiate contact -- that is handled automatically by MTP. When an application process closes an MTP connection, MTP sends a "goodbye" message to the destination node, and deletes the connection record. We may implement an "implied open", so that MTP will automatically open a connection whenever a process attempts to send a message to a destination host and port for which a connection has not yet been opened.

Receiving an MTP message is straightforward -- there is no necessity, as there is with TCP, for the receiving application process to parse the incoming data stream to find the boundaries between messages. Sending a message is also straightforward, but additional features are provided to allow the application process to monitor the status of the message so that it can be sure it has been delivered, and to kill it or change its priority, up until the ACK has been received. These features require that the sending application process be able to refer to a specific message it has already sent, and so MTP returns a unique message ID whenever an application process sends a message. The application process can also request to be informed when the message is ACKed, or when a timeout has occured.

Application processes can access data about a specific message, about the communications with a specific destination host, and about a specific communications link (within the limits of the network topology information provided to MTP in the connection opening process). Definition of what specific data should be provided, and how it should be aggregated to be most useful, will be on ongoing process. To acquire the necessary performance data, especially when communcations links are not reliable, may prompt the inclusion of additional mechanisms, such as packet transmission sequence numbers which uniquely identify each transmission event, rather than each unique message.

3.4 MSSMP Session Protocol

In addition to these Transport layer issues, but related to them because we use the same mechanisms both within and between physical platforms, is the need for Session Layer constructs (implicit or explicit) to cleanly support multiple sensor platforms and multiple control stations, and to permit the maintenance of platform control and the orderly transfer of control from one station to another in a variety of circumstances, including:

- one operator requesting, and receiving, transfer of control of a platform from another operator currently controlling it

- an operator assuming control of a platform which has lost contact with its current operator

- an operator (e.g., higher echelon commander) invoking a higher priority in order to "steal" control of a platform from its current operator

- split control, in which one operator acquires control of one subsystem (e.g., for problem diagnosis and repair) while another operator controls the rest of the platform

- data sharing, in which multiple operators may have "read only" access to a platforms sensor data output, while just one operator has actual control.

Development of these Session layer tools is just beginning.

4. CONCLUSIONS

The thrust of short range wireless communications systems developments in both the military and civilian sectors clearly indicate that Internet Protocol (IP) service will become a standard service model for tactical RF networks, and that data rates will be capable of supporting mobile robots and tactical security sensors. Our approach in the MSSMP project is to utilize a readily available COTS IP-based RF communications subsystem as a functional proxy for the tactical radio systems to be deployed in the next decade.

The effective utilization of COTS IP-based RF communications systems requires higher level (Transport layer and above) protocols capable of being tuned both to the performance characteristics (data and error rates) of the RF system and to the throughput and response requirements of the application. Transmission Control Protocol (TCP) effectively provides desirable communications transparency and robustness in an environment of high bandwidth and low error rate communications channels, but these same behavioral strategies work to the users' disadvantage in a communications environment of low bandwidth, moderate error rate channels and a network configuration potentially undergoing rapid changes. We are developing an appropriate communications management protocol layer that will run on top of the Internet standard User Datagram Protocol (UDP), allowing the nodes of a robotic security sensor system to effectively and efficiently provide the best possible balance of communications throughput and response time by adapting their communications behavior to the communications channel resources actually available. A key to this is to design the interface by which higher layer user processes access this Transport layer service so that user processes can assess the performance of the communications in real time.

ACKNOWLEDGEMENTS

This work has been performed under the Tactical Security Sensor Internetting and Integration (TSSII) Exploratory Development (6.2) project sponsored by the Defense Special Weapons Agency.

REFERENCES

1. T.A. Heath-Pastore, H.R. Everett, "Coordinated Control of Interior and Exterior Autonomous Platforms," *ISRA '94, Fifth International Symposium on Robotics and Manufacturing*, Maui, HI, August, 1994.

2. R.T. Laird, H.R. Everett, G.A. Gilbreath, "A Host Architecture for Multiple Robot Control," *Proceedings of the ANS Fifth Topical Meeting on Robotics and Remote Systems*, American Nuclear Society, Chicago, IL, (1993).

3. H.R. Everett, G.A. Gilbreath, T.A. Heath-Pastore, R.T. Laird, "Coordinated Control of Multiple Security Robots," *Proceedings of SPIE Mobile Robots VIII*, Boston, MA, September 1993.

4. Aironet Wireless Communications, Inc. *ARLAN 640 Ethernet Multipoint Bridge User Guide*, Revision A0, December 1994.

5. http://www.disa.mil/dii/execsum1.html

6. http://www.ado.army.mil

7. http://www.seas.upenn.edu/~ross/book/overview/protocol_stacks.htm

8. International Organization for Standards (ISO), "Information Processing Systems - Open Systems Interconnection - Basic Reference Model", ISO 7498:1984.

9. ftp://ds.internic.net/rfc/791.txt This and all other Internet Standards, called "Requests for Comments" or "RFCs", are available in the directory ftp://ds.internic.net/rfc/

10. ftp://ds.internic.net/rfc/rfc793.txt

11. ftp://ds.internic.net/rfc/rfc959.txt

12. ftp://ds.internic.net/rfc/rfc768.txt

13. ftp://ds.internic.net/rfc/rfc1889.txt; also see http://www.cis.ohio-state.edu/~cliu/ for a good discussion of RTP and related protocols.

14. Lin, J.C. and Paul, S. "RMTP: A Reliable Multicast Transport Protocol", *Proceedings of IEEE INFOCOM '96*, pp 1414-1424.

15. ftp://ds.internic.net/rfc/rfc908.txt

16. ftp://ds.internic.net/rfc/rfc1151.txt

17. Rome Laboratory Draft FY98 SBIR TOPIC #AF98-113, "Selective Retransmission Application Layer Protocol".

18. Wright, G.R. and Stevens, W.R. *TCP/IP Illustrated, Volume 2 The Implementation,* Addison-Wesley, Reading MA, 1994.

19. Feit, S. *TCP/IP: architecture, protocols, and implementations with IPv6 and IP security - 2nd ed*, McGraw Hill, New York NY, 1996.

20. Murphy, D.W., Bott, J.P., Bryan, W.D., Coleman, J.L., Gage, D.W., Nguyen, H.G., and Cheatham, M.P. "MSSMP: No Place to Hide", *Proceedings of AUVSI '97*, Baltimore MD, 3-4 June 1997.