Internetting tactical security sensor systems

Douglas W. Gage, W. Dale Bryan, Hoa G. Nguyen

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

ABSTRACT

The Multipurpose Surveillance and Security Mission Platform (MSSMP) is a distributed network of remote sensing packages and control stations, designed to provide a rapidly deployable, extended-range surveillance capability for a wide variety of military security operations and other tactical missions. The baseline MSSMP sensor suite consists of a pan/tilt unit with video and FLIR cameras and laser rangefinder. With an additional radio transceiver, MSSMP can also function as a gateway between existing security/surveillance sensor systems such as TASS, TRSS, and IREMBASS, and IP-based networks, to support the timely distribution of both threat detection and threat assessment information.

The MSSMP system makes maximum use of Commercial Off The Shelf (COTS) components for sensing, processing, and communications, and of both established and emerging standard communications networking protocols and system integration techniques. Its use of IP-based protocols allows it to freely interoperate with the Internet -- providing geographic transparency, facilitating development, and allowing fully distributed demonstration capability -- and prepares it for integration with the IP-based tactical radio networks that will evolve in the next decade.

Unfortunately, the Internet's standard Transport layer protocol, TCP, is poorly matched to the requirements of security sensors and other quasi-autonomous systems in being oriented to conveying a continuous data stream, rather than discrete messages. Also, its canonical "socket" interface both conceals short losses of communications connectivity and simply gives up and forces the Application layer software to deal with longer losses. For MSSMP, 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 being developed to support the effective transfer of control of multiple platforms among multiple control stations.

Keywords: security sensors, surveillance, internetworking, gateway, TCP/IP, UDP, reliable transport, datagram

1. INTRODUCTION

1.1 The Multipurpose Security and Surveillance Mission Platform (MSSMP)

The Multipurpose Security and Surveillance Mission Platform (MSSMP)[1, 2], initiated in 1992 as the Air-Mobile Ground Security and Surveillance System (AMGSSS)[3, 4], 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. While the MSSMP system consists of three air-mobile remote sensing packages and a base station, MSSMP sensor packages may also be mounted on manned or unmanned ground-mobile platforms, or deployed as man-portable stand-alone units (Man-Portable Networked Sensor System, or MPNSS[5, 6]).

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.

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 value 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[7] as the basis for communications between all remote payload subsystems and the control/display station.

1.2 IP-based RF networks for security sensors and other military systems

COTS wireless Ethernet transceiver and bridge technologies such as the Arlan devices adopted by MSSMP provide a number of features which are very attractive for tactical security sensor systems, including self-organizing networking, data packet autorouting, mobile roaming/handoff, and remote configuration control using Simple Network Management Protocol (SNMP). 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.[7]

A number of military programs, including DARPA’s Warfighter Internet and Global Mobile Information Systems (GloMo) programs, 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*[8], 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[9], calls for the establishment of a seamless communication infrastructure called the “Warfighter Information Network” 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 Layered Protocols and the Internet

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. A key notion is that of protocol *layering*: dividing the complete set of agreements between two communicating systems into a number of independent hierarchical layers, with "protocol entities" at each layer making use of services provided by the interaction of the protocol entities at the next lower layer. The Open Systems Interconnection (OSI) Reference Model[10], developed by the International Standards Organization (ISO) in the early 1980s, and intended to be general enough to serve as the basis for description of even the most complicated network systems, 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. 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 are implemented either explicitly or implicitly, are subsumed into the Application layer in the Internet protocol model.

2.2 Standard Internet Transport Layer Protocols

The Internet's Network layer protocol is named Internet Protocol (IP),[11] and so this protocol layer and its associated processing is often refered to as the "IP layer". IP is 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[12], 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.

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[13], is less than 3 pages long. UDP is supported as part of all standard TCP/IP software packages.

Use of the TCP/IP protocol suite has permitted the Internet to grow rapidly, steadily, and nondisruptively over many orders of magnitude in the past decade and a half. But military tactical RF communications capabilities -- 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 -- are very different from the extremely high bandwidth permanently-installed communications channels that support the Internet. This work addresses some of the issues associated with using limited-performance IP networks to support tactical security/surveillance sensor applications -- in other words, it seeks to determine the best ways for tactical 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 Excessive (and Inappropriate) Use of TCP

If we choose to categorize communications services as either conveying a data stream (like a telephone) or sequence of discrete messages (like regular postal mail), then TCP provides a stream service, while UDP provides a message service. However, a number of important Internet applications that require the transport of messages or files -- including email, World Wide Web, and FTP -- use an underlying byte-stream oriented TCP connection. One wonders why a reliable message-oriented protocol isn't used instead. 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 and very widely used than to develop something new that was better tuned to their needs. This is not to say that other protocols have not been developed for any number of different application niches, such as RTP (Real Time Protocol[14]) and RMTP (Reliable Multicast Transport Protocol[15]), but that no general purpose protocol for the reliable transmission of message-oriented traffic has gained wide acceptance.

The fate of many of these development efforts is typified by that of Reliable Data Protocol (RDP)[16, 17]. 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. It therefore provides 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), RDP is an "Experimental Protocol" whose implementation status has remained "Limited Use".

2.4 TCP Performance Limitations

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

TCP's basic goal is to convey data to its destination as expeditiously as possible, but 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 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, 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. RFC 1323 and RFC 2001 are illustrative.

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. 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 used 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 argue below that the higher levels of control of a deployed security sensor platform or mobile robot or other continuously behaving system *need* to know some of what's actually happening in the communications processes supporting it. 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. MSSMP TRANSPORT LAYER PROTOCOL

3.1 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.[4]

- 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.

3.2 MSSMP Transport Layer Mechanisms

Ideally, we would like to develop a tightly written, efficient, reliable *message-based* Transport protocol layer, which implies a system for buffering the messages to be transmitted, message-oriented (as opposed to TCP's byte-oriented) sequence numbers, retransmission timeouts, and positive and/or negative acknowledgements. Furthermore, we would like to include all 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", including selective acknowledgements. And we would like to provide a Transport layer service interface that allows us to address the concerns and goals discussed in Section 2.5 and Section 3.1 above.

Our implementation approach is to follow the well-trodden path described by Stevens for transaction-handling applications:

> Today the choice an application designer has is TCP or UDP. TCP provides too many features for transactions, and UDP doesn't provide enough. Usually the application is built using UDP (to avoid the overhead of TCP connections) but many of the desirable features (dynamic timeout and retransmission, congestion avoidance, etc.) are placed into the application, where they're reinvented over and over again.[20]

We will, however, implement only minimally the usual "desirable features", and instead focus on developing and using a generalized "socketlike" interface that will allow the Application layer to have read-only access to data which would otherwise be completely internal to the Transport layer 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 our protocol (let's call it MSSMP Transport Protocol, "MTP" for short) in a default mode with having to worry about the generalizations.

When a user process opens a connection to another node, 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 periodically attempts 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 also 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, and software implementation is greatly simplified, since 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 for the application process. The only difference from the usual socket interface is that the MTP send or write command returns a unique message ID whenever an application process sends a message, instead of returning the number of characters written. This message ID allows 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. The application process can also request to be informed when the message is ACKed, or when a timeout has occured.

Of course, the actual implementation of reliable message transmission and reception on top of UDP will require the implementation of message sequence numbers, retransmission timeout timers, acknowledgements, and so forth. The initial implementation of this infrastructure will be as simple as possible, since the underlying Ethernet link layer already provides low-level adaptive packet retransmission, and we wish to focus our efforts on exploration of the unique features of our scheme.

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 an ongoing process. To acquire the necessary performance data, especially when communications 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.

4. MSSMP SESSION LAYER PROTOCOL

ISO Standard 8326, *Connection Oriented Session Service Definition*,[21] defines connection-oriented services that establish sessions and negotiate session parameters. Operations are divided into the session conection establishment phase, data-transfer phase, and session connection release phase. The MSSMP system requires Session Layer mechanisms (implicit or explicit) to cleanly support multiple sensor platforms and multiple control stations, permitting the orderly transfer of control of a platform 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.

Of course, these requirements are not unique to MSSMP, but apply to any system with dynamic client-server relationships and requirements for uniqueness of control. The initial MSSMP Session Layer implementation does not deal with several additional possible complications:

- command interruption, which has implications for safety and system integrity; the concept of "emergency override" is one which must be further considered

- the concept of "delegation" of precedence

- assignment of operator precedence -- if an operator requires precedence strictly higher than a current controller to preempt it, then a race ensues between two operators of equal precedence to see who takes control first.

In general, the problem of preventing both deadlocks (the state where each client holds enough privileges to prevent each other client from being able to do anything, but not enough to do anything itself) and thrashing (a "fight" between multiple clients each of whom has enough privileges to preempt each other) must always be kept in mind.

4.1 MSSMP Commands and Command Execution Process

The implementation of the required Session Layer mechanisms must be fully integrated into the MSSMP platform's command structure and command execution process.

The protocol between the MSSMP remote platform and the operator CD is essentially that between a server and a client. The MSSMP platform's Payload Processor (PP) provides the remote operator with the ability to control the platform's sensor resources: TV camera, FLIR camera, laser rangefinder, pan/tilt unit, and optional acoustic sensing system. It does this by accepting and executing user commands from the operator CD, returning system and subsystem status messages to the CD, and coordinating the acquisition and flow of image data back to the CD. Command execution requires communication between the PP and the sensor subsystems via RS-232 serial streams (FLIR, laser rangefinder, pan/tilt, acoustic subsystem) and/or discrete signals (TV focus and zoom) via a relay board.

High Level (View) Commands. The operator at the CD sees payload functionality principally in terms of images being displayed at the CD after they have been acquired and transmitted by the platform. The CD passes an image specification (or *view*) to the PP, and the PP acquires and returns the image as soon as it can. A view specifies: the direction the pan/tilt unit is to point, whether the TV or the FLIR camera is to be used, how the camera is to be configured (focus and zoom values for TV; image polarity, level and gain for FLIR), and how the acquired image is to be processed by the Image Processor (image compression ratio, contrast enhancement, histogram equalization).

Alert Response Specification. The Image Processor (IP) -- which may be realized as a process on the PP computer hardware, or actually exist as a separate computer connected to the PP via Ethernet -- provides a motion detection function, which is invoked with a view command. The operator can also specify whether the platform should automatically return an image and/or a laser range when motion is detected. In addition, MSSMP has been tested with an acoustic detection subsystem which sends the PP a report of acoustic detections every 1.25 seconds, each acoustic source being labeled with its azimuth from the MPP, its type (ground vehicle, propeller aircraft, jet, aircraft. helicopter, or unknown), and a measure of the acoustic subsystem's confidence in the detection. The CD operator can install filters in the PP to specify which acoustic detections will be forwarded to the CD, and also specify whether to automatically return an image and/or a laser range.

Low Level Commands. An extensive set of low level commands allows the user to exercise detailed control of the platform's subsystems, e.g.: set the pan/tilt's tilt to -30 degrees, set the TV zoom to 20, increase the FLIR gain by 35. Besides providing the operator with "hands-on" control, these commands support system integration and debugging, and troubleshooting in the field.

Command Programs. Both high level and low level commands can be strung together in command programs up to hundreds of steps long. Programs are usually used to acquire a panorama of images, or to repeatedly (via the special command REPEAT) scan a number of potential threat directions with motion detection. Programs are downloaded from the CD, and can also be uploaded back to the CD for inspection and validation. The PP can store 3 programs, but only one program can be executing at any one time; the operator controls program execution with the PP commands RUN, HALT, and CONTINUE.

Command Execution Process. Many platform subsystem functions take a very long time when measured in terms of processor execution cycles. For example, a full pan excursion takes almost 2 seconds, a full range camera lens zoom excursion takes 6 seconds, and the FLIR takes several minutes to initially cool to its operating temperature. Hence the PP software must be written so that long-duration commands do not tie up the PP's CPU, which is running plain old single-threaded DOS as its "operating system". Moreover, operations of different subsystems must be allowed to overlap, so that the pan/tilt can be panning while the camera lens is zooming and so forth. On the other hand, high level view commands require that subsystem operations be properly sequenced to succeed. For example, in response to a view command the PP must be able to command the pan/tilt to its new position and then immediately command the camera zoom and focus to their new values, then, when both operations have completed, wait an additional 2 seconds for the camera auto-iris to stabilize, and only then command the IP to grab and process the image.

To support this flexibility of operation, a resource lock mechanism has been implemented which permits the simultaneous execution of multiple commands not requiring the same resources, while forcing commands that require resources already in use to wait until those resources become free. Resource granularity is at the subsystem level, i.e., pan/tilt, TV, FLIR, laser rangefinder, and image processor.

4.2 MSSMP Session Layer Mechanisms

The MSSMP Session Layer extends this existing resource lock mechanism. Each command has an associated list of the resources required for its execution. In addition, each resource has associated with it the name of the single client that currently controls the resource, plus lists of the (potentially multiple) clients that should receive (1) data, (2) command, response, and status info, and (3) error notifications for the subsystem.

When a command is received from a client, the PP checks to see if that client is listed as the controller for each of the resources that the command requires. If it is not, a response is sent back to the client telling which required resource control privileges are missing, and identifying which other clients (if any) currently hold them. If the client has all needed control privileges, then the PP checks to see whether any of the required resources are already in use by other commands. If so, the command is queued for execution when the resource becomes free. If not, it is executed immediately. (Of course, the CD also checks the operator's command entries, and generally does not permit the operator to send a command for which he does not already have the required control privileges.)

Command responses, status info, and data are also labeled (internally to the PP) with the resources that produce them, so that they can be sent to all the clients on the corresponding client lists associated with these resources.

The process of logging an operator at a CD (client) onto a sensor platform (server) requires specifying the desired privilege -- control, data, command/response/status, and/or error notification -- for each of the resources on the platform. In most cases, the human operator need explicitly specify only "control" or "data" for the entire platform -- the CD software handles the details behind the scenes. A request for control of a resource that does not currently have an assigned controller is immediately granted. If a requested resource already has a currently assigned controller, a message is sent to that controller requesting that it relinquish control. The current controller may explicitly relinquish control, or refuse to do so. If the request message is not acknowledged within a reasonable timeout period (i.e., communications are "broken") then control is transfered anyway. Each client CD is assigned a precedence appropriate to the operator's rank and role in the overall deployment, and a higher precedence operator can choose to preempt a lower precedence operator's control of a resource (but must do so explicitly).

5. 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 to 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 measure 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. D. W. Murphy, J. P. Bott, W. D. Bryan, J. L. Coleman, D. W. Gage, H. G. Nguyen, and M. P. Cheatham, "MSSMP: No Place to Hide," *AUVSI '97 Proceedings*, Baltimore MD, 3-6 June 1997, pp. 281-290,.

2. "Multipurpose Security and Surveillance Mission Platform," http://www.spawar.navy.mil/robots/air/amgsss/mssmp.html.

3. D. W. Murphy and J. P. Bott, "On the Lookout: The Air Mobile Ground Security and Surveillance System (AMGSSS) Has Arrived," *Unmanned Systems*, Vol. 13, No. 4, Fall 1995.

4. H. G. Nguyen, W. C. Marsh, and W. D. Bryan, "Virtual Systems: Aspects of the Air-Mobile Ground Security and Surveillance System Prototype," *Unmanned Systems*, Vol. 14, No. 1, Winter 1996.

5. W. D. Bryan, H. G. Nguyen, and D. W. Gage, "Man-Portable Networked Sensor System," *SPIE Vol. 3394: Sensor Technology for Soldier Systems*, Orlando, FL, 15-17 April 1998.

6. "Man-Portable Networked Sensor System," at World-Wide-Web URL http://marlin.spawar.navy.mil/D37/.

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

8. "Command and Control Umbrella," http://www.ado.army.mil/smrtbook/sbpage8.htm.

9. "The Army Digitization Office," http://www.ado.army.mil.

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

11. "Internet Protocol," ftp://ds.internic.net/rfc/rfc791.txt This and all other Internet Standards, called "Requests for Comments" or "RFCs", are available in the directory ftp://ds.internic.net/rfc/.

12. "Transmission Control Protocol," ftp://ds.internic.net/rfc/rfc793.txt.

13. "User Datagram Protocol," ftp://ds.internic.net/rfc/rfc768.txt.

14. "Real Time Protocol," tp://ds.internic.net/rfc/rfc1889.txt; also see http://www.cis.ohio-state.edu/~cliu/ for a good discussion of RTP and related protocols.

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

16. "Reliable Datagram Protocol," ftp://ds.internic.net/rfc/rfc908.txt.

17. "Reliable Datagram Protocol," ftp://ds.internic.net/rfc/rfc1151.txt.

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

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

20. W. R. Stevens, *TCP/IP Illustrated, Volume 1 The Protocols,* Addison-Wesley, Reading MA, 1994, p 352.

21. International Organization for Standards (ISO), *Information Processing Systems - Open Systems Interconnection - Connection Oriented Session Service Definition,* ISO 8326: 1987.

22. D. W. Gage, "Network Protocols for Mobile Robot Systems", SPIE Vol. 3210, *Mobile Robots XII*, Pittsburgh PA, October 1997, pp 107-118.