GROUND STATION VIRTUALIZATION

James Cutler

Stanford University
jwc@stanford.edu

ABSTRACT

We apply the end-to-end principle, which states that lower layers of systems should support the widest possible variety of services and functions, to ground station systems. We isolate core ground station services and discuss a mechanism for flexible application level support. Taking into account space system trends such as adoption of terrestrial Internet protocols, we develop a reference model that captures core ground services. In an effort to promote end user flexibility, we organize these services into hierarchical layers that allow low-level hardware commanding, automation of contact sessions, and peer-to-peer station cooperation across multiple space missions. Given this model, we describe our reference architecture and its implementation based on XML messaging and standard Internet software systems built with recovery-oriented design principles. We also discuss a testbed which consists of globally distributed, university ground station systems.

1. INTRODUCTION

Trends in spacecraft operation are moving towards increased satellite connectivity that will enable 24x7, end-to-end access between users and satellite data. Some even foresee and hope that communication will soon be eliminated as a constraint for accessing space-based information[10].

Focusing on similar trends and performance issues, the terrestrial computing research community has to a small degree reached this goal of eliminating communication as a constraint. They have increased computing power by 10,000-fold in the past two decades and provided a ubiquitous networking environment, a network of networks, called the Internet.

A strength of Internet systems has been the rigorous use of the end-to-end design principle which states that lower layers of systems should support the widest possible variety of services and functions[32]. In other words, the application knows best how to perform its task so lower layers should provide common, widely-used building blocks. The simple core Internet protocol has given end users incredible flexibility to design diverse applications ranging from email, to file trading, to voice over IP, and to streaming video.

This increase in capability has come at a price though. As these computing increase in complexity, they become brittle and fragile to operate where downtime can cost hundreds of thousands of dollars per hour[28]. In order to reduce the impact of these inevitable failures, research is now focusing on systems that recovery quickly[16].

Given these trends, the goal of our research is two-fold: to develop infrastructure to make space-based information more accessible with a wider impact, and to make this infrastructure robust and reliable while being built from unreliable components. We are applying end-to-end principles to ground station systems to highlight core, widely-used services and create more flexible architectures for end user applications. We are also developing recovery-oriented computing principles, where we embrace failure as a fact and seek to build systems that recovery quickly with minimal side affects. We are applying these principles to a network of ground stations that support university satellite missions. Our work is the partnership of two laboratories at Stanford University: the Software Infrastructures Group (SWIG) that has developed technology for robust Internet infrastructure, and the Space Systems Development Laboratory (SSDL) that builds small, low-cost research micro and nano satellites.

This paper further describes our efforts to apply these two principles to ground station systems. First, we describe the end-to-end principle in more detail and possible applications to ground stations. Then, we address the needs of future ground station systems by extending current ground station architectural reference models. Next, we organize this model into hierarchical levels based on autonomy. We then describe important quality attributes of ground station architectures and how these were implemented in our reference architecture. We conclude with comments on cost benefits, future and related work.

2. END-TO-END PRINCIPLES

The trends in space system development have been noted by many. There is a need for drastic increase in access opportunities to space assets[7]. End-to-end data transfers are needed between space and ground assets as well direct communication between orbiting space assets[11]. Ground systems must support multiple missions from a growing num-

This work is supported by NASA grant NAG3-2579.

ber of different institutions. Operations teams are moving towards higher levels of mission autonomy. Future ground systems must meet these new trends as well support existing, legacy space missions.

Concerning these trends, many similarities exist in Internet systems. Efforts are under way to provide pervasive computing and ubiquitous networking to enable even the smallest embedded devices to share information over the Internet any time and anywhere[36]. Wired and wireless networks provide networking for a range of heterogeneous computing platforms with applications ranging from text messaging to video streaming. With some success due to technologies such as the Internet protocols, Ethernet, and IEEE 802.11b, terrestrial networks are achieving the removal of communication as a constraint. Given this success, it's important to take note of Internet design principles and their application to ground systems.

A key enabler to the rapid success and proliferation of Internet connectivity and services has been the fertile ground for innovation offered by the Internet. For example, what has enabled graduate students in their spare time to create the likes of Yahoo! and Google? Maybe it was just soft advisors being easy on their students, but some have attributed this fertility to the simplicity of the core Internet protocol and the flexibility given to the end user to develop applications[20].

At the heart of this is the end-to-end argument, which states that the function in question can completely and correctly be implemented only with the knowledge and help of the application standing at the end points of the communication system[32]. Therefore, "a lower layer of a system should support the widest possible variety of services and functions, so as to permit applications that cannot be anticipated". Building complexity into a network often optimizes it for that particular application to the detriment of other unforeseen applications[31].

There are numerous example applications of this principle in the computer systems community. Consider the paradigm shift in computer architectures that took place in the eighties where complex instruction set computers with high level hardware support for programming languages were replaced with simpler architectures, called reduced instruction set computers (RISC). RISC systems provided basic instruction set primitives and reported two to fives times better performance than complex architectures on the same silicon wafer technology. These reduced instructions enabled compilers to optimize the architecture for multiple programming languages rather than just the specific one supported by hardware. RISC processors now dominate the microprocessor market[29].

Consider the flexibility of the Secure Socket Layer (SSL) [17]. The SSL protocol enables client/server authentication and message encryption for web applications. Rather than specify apriori a specific encryption technology, SSL supports client/server negotiation of available cypher suites. They have standardized a protocol to enable flexibility in choice of encryption technology. Thus, legacy as well as future technologies are supported.

Again consider the flexibility of the Hypertext Transfer Protocol (HTTP). The original CERN developers created it to simply share documents over the network[18]. Application developers are now using HTTP to trade stocks and facilitate financial transactions, perform remote method invocation using technologies such as SOAP[35], and even create tunnels that run SSH over HTTP.

In contrast, ground stations systems have typically been complex and optimized for single missions rather than multiple missions. Some of this is due to the inherent complexity of space to ground communication. However, consider that these stations are tackling low level physical link parameters as well as application level processing such as telemetry parsing and archiving. The difficulties in cross mission and agency standardization efforts have manifested themselves as complexity in the ground station. It's not uncommon for missions to supply equipment at station installations [38, 37].

Given the success of the Internet as described above and the complexity of current ground stations, can we apply the end-to-end argument to ground systems and provide the same fertile ground for innovation in ground system development and the same levels of interconnectivity? Can we standardize basic ground station services and provide a standard mechanism for flexible application level support? We begin to tackle these questions in this paper.

3. CONTEXT

The fundamental purpose of a ground station (GS) is to enable communication between ground users and space assets. In addition to the RF equipment needed for communication, ground stations traditionally have a suite of support systems for mission-specific data handling needs such as demultiplexing of data streams, encryption functions, data compression, time tagging, data storage, data quality measurements, and spacecraft ranging. This has led to challenges in multi-mission support due to highly specialized mission-specific equipment and a lack of flexibility for the end users [37].

With the adoption of terrestrial networking standards for end-to-end communication between space and ground systems, the core function of a ground station is simplifying and becoming similar to that of a standard Internet router. This adoption is evident through recent space system work [22][24][21]. Therefore, the fundamental purpose of a ground station is evolving to become more simple; it is to bridge space and terrestrial networks and route packets appropriately.

Despite these similarities with Internet routers, there are some basic differences between them and ground stations. These differences must be addressed when architecting future ground stations.

Due to the tracking constraints and narrow beam widths of the RF links [1], communication channels tend to be circuit switched; a ground station is scheduled for a particular time interval to exclusively maintain a communication channel with a single satellite. The bidirectional pointing requirements for high speed communicational channels and slow antenna slew rates prevent rapid multiplexing between multiple satellites. In contrast, the physical networks for routers are fixed and don't require reconfiguration for each packet stream. Therefore, while ground station resources remain scarce, this circuit-switched induced bottleneck requires efficient scheduling to enable multiple mission support.

Ground stations often support large downlink missions (hundreds of megabits and soon to be gigabits per second) at the edge of terrestrial networks were bandwidth is limited. Data must be stored and forwarded at a later time, sometimes by sneakernet[33], and this potentially requires application level knowledge. In contrast, routers tend to be closer to the Internet core where bandwidth is high. The packet life time in router is near zero, where they are quickly forwarded or dropped.

Internet routers contain no knowledge of the application. However, ground stations supporting legacy missions often have large amounts of application knowledge. For example, they sometimes demultiplex data streams and tag data with accurate time stamps. In the near future, ground stations must flexibly support these legacy missions. However, for the example given, this high level of application support will diminish as missions begin using ground compatible packet protocols and space-based clocks (such as those based on GPS).

The complexity of ground stations is currently significantly greater than Internet routers. In fact, ground stations usually contain a router in addition to all the equipment needed to support ground to space communication links. This increased complexity lowers the mean time to failure and potentially increases the mean time to repair. Hardware repairs are manually intensive and regular inspections are needed[14]. To bound costs, architectures must manage this complexity by decreasing failure detection and recovery times.

Also, ground stations require real time (or near real time) control of resources to maintain satellite contact channels. Telemetry feedback is used to adjust antenna pointing angles and receiver frequencies to maximize received signal strength. In contrast, routers simply manage queues without the complexity of hardware feedback control.

Any application of successful Internet design principles must take these differences into accounts. Given the rising software presence in ground systems[14, 38] and the distributed nature of a ground station network, the similarities are large enough to explore the cross fertilization of principles.

4. HIERARCHICAL REFERENCE MODEL

We begin with an attempt to capture core ground station services, ones that are shared across multiple missions, and separate out mission specific services. We divide station services along autonomy lines into three hierarchical layers which permit low level hardware control, contact automation for a single station, and peer cooperation among ground stations to enhance space to ground links. We layer these services to hide information[25]; the encapsulation of module specific functions to hide device heterogeneity and provide common, highly utilized virtualized services. Our eventual goal is to develop an architecture that offers core services and standardized mechanisms for flexible application level services. [2]

The *virtual hardware level* captures the fundamental capabilities of low-level ground station components and enables generic commanding of heterogeneous hardware. This is a master/slave control paradigm where the ground station exposes lower level control interfaces for all hardware. At the heart of the ground station is the dedicated hardware to support ground to space communication links, which we call *hardware pipelines*.[3]. Pipelines convert space radio transmissions into digital bits that can be placed on digital networks (and vice versa). These consist of hardware associated with antennas, low noise receive amplifiers (LNA), output power amplifiers, radios, modems, and multiplexing hardware for flexible configurations. Pipelines also perform ranging functions to measure satellite distances and positions. Others have provided physical views at this layer[37, 38, 14].

The *session level* captures typical automation tasks and services of a single ground station installation. Users define a session, which tasks these automation services over

[1] Beams are narrowing even more has bit rates increase, especially when optical links become common place. Phased array antenna technology may soon allow multiple RF links to be maintained through a single antenna system, but stations are likely to remain circuited switched.

[2] On an aside, it's interesting to note many of the functional similarities between ground stations and satellites. Except for the payloads, satellites have similar functions. There is some interesting synergy between the two. For example, our Stanford-built satellites and ground stations both run Linux on Intel-based processors. We even use some of the same embedded technology as well to control power distribution and signal multiplexing. Also, with the rise of formation flying missions, and if ground stations support some level of application level services, the stations appears to be just another node in the formation, a ground node.

[3] These have been called "chains" and "strings" in [38]. We use pipeline to keep our work consistent.

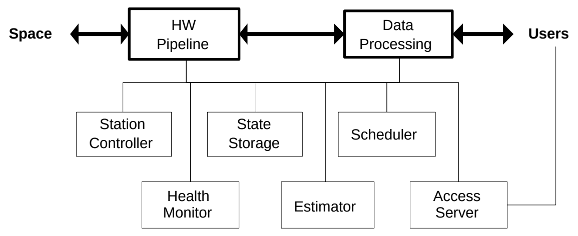

Figure 1. Component view at the session level.

a specific time interval to maintain communication channels. A communication channel definition includes one or more pipelines (transmit, receive, or transceive) and associated data processing services. Session level services employ the virtual hardware level primitives to control lower level ground station resources. A functional view of a ground station at the session level is found in figure 1.

The *scheduling service* accepts reservations for the use of ground station systems. Ground requests are processed through terrestrial networks. Space requests are received through low resource utilization pipelines (such as simple omnidirectional antennas). Until stations are no longer a scarce resource, scheduling will be critical function. Resource availability and user access priority are taken into account during the scheduling of sessions.

The *station controller* automates execution of satellite contact sessions. It monitors scheduled contact session requests, it configures ground station hardware to support the requested communication channels, and enables requested automation and data processing services. It performs real time control of station hardware to maintain and optimize communication channels during a satellite pass. Antennas are tracked, radios tuned, and link parameters (such as FEC levels and bit rates) adjusted to account for variable bit error rates. The controller also manages health monitor output and recovers from failures automatically or alerts on non-recoverable errors.

The station is monitored for proper operation by *health monitor* systems. These consist of sensors, both software and hardware, logging resources for storing telemetry, and health assessment functions for detecting failures and performance degradations.

The *estimator* determines the target satellite position, antenna pointing angles, and Doppler correction factors. Possible sources for this information include pipeline ranging systems, satellite provided GPS data, or calculated values from orbital element sets.

The *state management service* stores session descriptions and their schedules, session products which contain GS telemetry and communication channel bit logs, a cache of satellite configuration information (pipeline frequencies, data processing needs, etc.), user access information, etc.

The *remote access server* authenticates remote users and provides secure, encrypted ground station control. It controls access to the scheduling services, the data processing services, and also enables access to communication channels and low level virtual hardware commanding.

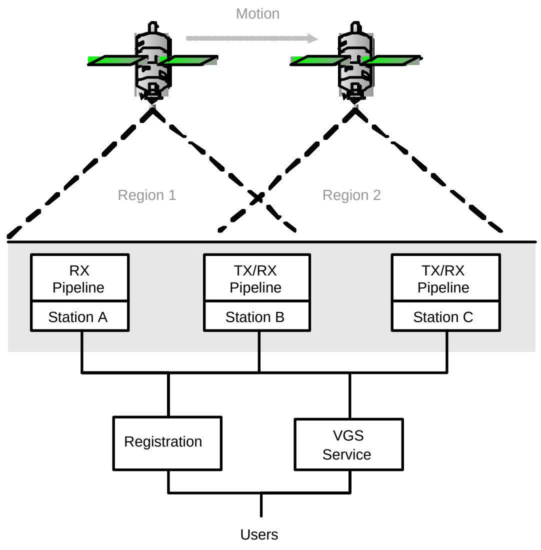

Figure 2. Here is a component view at the network level. In region 1, station A and B cooperate on maintaining the contact. Station A's transmit power amplifi er has failed and B has taken over transmit responsibilities. As the satellite moves into region 2, station B takes over the receive functions to provide a seamless transition. Station B and C then coordinate to provide an optimized virtual station, where B is the primary receiver because of its higher gain antennas and better terrestrial network access and C continues with transmitting until the satellite is out of range.

Processing bits on the ground side of the pipeline is handled by the *data processing* services. Networking and communication services include bit synchronization, forward error correction (FEC), and link and network level protocol management. At the application level, there are endless possibilities of data services, some of which have been mentioned above. In our architectural development, we hope to separate out these services and replace them with a mechanism to allow missions to run custom services without the ground station having explicit knowledge of them.

The *network level* captures the services of a ground station network.[4] These networks provided increased capabilities over single ground station installations and are federated from stations under different administrative domains [12]. Link intermittency is reduced and temporal coverage is increased as globally distributed ground stations with

[4]In the past we have called this the "mission level" but now feel "network" is more appropriate. In the future a mission level may be added to include services common to satellite operations. This broadens the scope of this work from ground stations to ground systems.

overlapping contact windows handoff satellite contact session. Local clustering [5] of ground station teams enables static and dynamic optimization of a ground station system through composition of a virtual ground station, a ground station composed of components and services distributed across multiple installations. See figure 2 for views of these networks.

The *registry service* accepts registrations from ground stations offering their capabilities to the network. This registry service enables users to locate available ground station resources. Full scheduling authority rests with the ground stations.

The *virtual ground station service* composes distributed ground stations and presents a virtual single interface to station end users. This service enables peer ground station cooperation and masks failovers and handoffs without explicit user knowledge. Autonomy can be handled centrally with stations acting as slaves or distributed among them where they act as peers.

These three layers capture the core services of a ground station network and present different interfaces to users based on autonomy levels. At the virtual hardware level, users have full control of station hardware. At the session level, a station is tasked to automatically track a satellite and provide network data connections between ground and space assets. At the network level, teams of stations are tasked to maintain extended contacts, to optimize links, and provide redundancy in the presence of failures.

5. SYSTEM QUALITY ATTRIBUTES

We have already discussed the application of the end-to-end argument in the context of ground station architectures. This has lead us to seek simplification in core ground station resources in an effort to promote flexibility and interoperability in ground station systems. Eventually, core ground station services will be standardized along with mechanisms for flexible, application-specific services to be run at ground stations. These are two of the three quality attributes we focusing on.

In the past, ground stations have been highly optimized for performance; they have been stove-piped solutions to meet the needs of a particular mission. Similar, computer system researchers have focused on performance, increasing computing power 10,000 fold in the last twenty years[16]. However, this has resulted in large distributed computing systems that tend to be brittle and difficult to repair. Do ground stations suffer from the same complexity-induced difficulties?

It has been noted that the drive for lower-cost ground stations is reducing the redundancy in the these systems, thereby increasing single points of failure and failure rates[5]. Hardware failures tend to dominate ground station failures, and the rise of software intensive components will likely lead to more software failures as well[14, 38]. So, it appears that ground stations are also suffering from reliability issues as they increase their performance and complexity.

Given this reliability issue, a primary attribute of our system architecture is availability. Our method is based upon recovery-oriented computing principles (ROC) [27]. We assume failures due to people, hardware, and software are facts that must be coped with over time. Despite decreases in failures rates, they still occur as our systems become more complex. ROC advocates increased design emphasis on recovery rather than exclusively fault avoidance. In some cases, systems that recovery quickly are more valuable then systems that don't fail often (especially if they take long to recover)[15].

With the increasing lines of code in ground stations, software failures and errors will increase. We have recently applied one of the ROC techniques, recursive restartability [9], with some success to ground stations systems[8]. Concerning hardware failures, we assume most ground stations have little redundancy (as noted by[5]). Thus, we are using redundancy in the network of ground stations to recover from single point hardware failures[12].

Tradeoffs are a necessary function of architectural development. Traditionally, space systems have been optimized for performance while trading off flexibility and interoperability. However, the rising total cost of ownership of space systems is forcing us to re-evaluate these two neglected qualities. Also, requirements for lower cost systems is resulting in degrading system reliability. Therefore, we are shifting our focus to develop new architectures and systems that meet these challenges for robust systems that are flexible enough to interoperate with multiple missions.

6. MERCURY REFERENCE ARCHITECTURE

Given this hierarchical reference model, we have developed a ground station architecture and a reference implementation, called Mercury, to support university satellite missions. The latest version of our Mercury system, version 1.2.0, combines a three-tiered web architecture (Linux, Apache, MySQL, and PHP) and a suite of loosely coupled control software components (written primarily in Java) to implement the virtual hardware and session levels of the reference model. Development of the network level is underway.

We are partnering with universities around the world to build a global ground station network that supports university satellite missions. These satellite missions tend to be experimental research satellites, testing new engineering technologies in the space environment or performing basic science missions[13, 34]. They have limited financial

[5] Local refers to stations simultaneously within the footprint of a satellite and depends on satellite altitude.

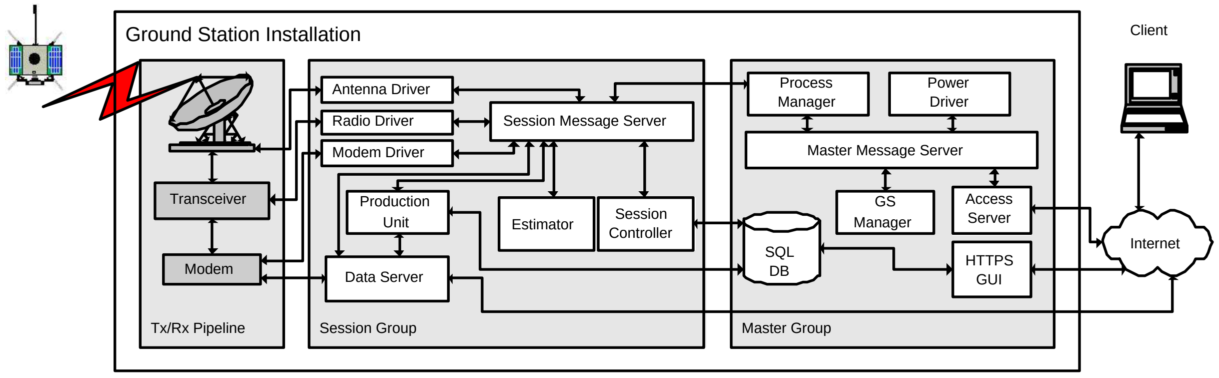

Figure 3. Software component view of the Mercury system with only several pipeline hardware components. Computers and other hardware are not shown. Internet security of the system will be discussed in a future work.

resources and are often built from non-space rated, COTS components. Though their communication systems tend to be low-rate (less than 38.4Kbps) due to financial constraints, these missions still are fertile ground for innovation and experimentation in space operations. For example, QuakeSat, a small nanosatellite built by students at Stanford, will be running end-to-end Internet protocols for their primary command and control[23].

At the heart of the Mercury is an XML messaging system for distributing command and control messages. We have captured the hierarchical reference model in the Ground Station Markup Language (GSML)[3]. GSML enables human and software agent-based control of ground station resources and serves as a testbed for exploring generic commanding of heterogeneous ground station components. The control software components are loosely coupled through a centralized messaging server. The server supports communication links such as TCP/IP, UDP/IP, and RS-232 serial links. We have developed a specification file that captures the GSML primitives. We compile this to automatically generate code that hides low level communication issues (such as TCP port opening and closing), and provides a GSML application programmer interface (API) to code writers. Many of our messaging principles are described in[30].

Figure 3 shows a software component view of the Mercury architecture. The *master group* of software components manages station configuration and scheduling of contact sessions. It also contains any hardware drivers that are independent of sessions or span multiple sessions. Stable storage of configuration and scheduling data is in the SQL database. Scheduling of ground station resources is through an HTML interface for human users and a GSML-enabled access server for software agents. The ground station manager monitors scheduled sessions and enables execution of contacts at the appropriate time. The process manager monitors all software components in the ground station and attempts recovery on failures.

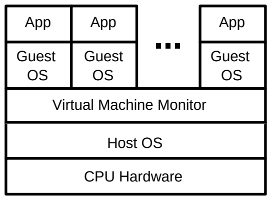

Figure 4. A virtual machine systems with multiple guest operating systems running on a single host.

The *session group* of software components provides command and control of ground station resources to maintain communication channels. Loosely coupled hardware drivers are controlled by the session controller. The estimator provides pointing angles and frequencies for the passing satellite. For non IP packets, the data server interfaces to the modems and encapsulates data in IP packets for transportation across terrestrial networks (we currently support AX.25 and IP). The production unit logs GS telemetry as well as bytes transmitted to and received from the pipelines. These are archived in the database.

We have discussed virtualization of ground station specific resources. We are also taking advantage of the virtualization of computing platforms. All ground station processes run inside of virtual machines (VM). We use VMware [1] to run multiple guest operating systems on a single host operating system and platform, see figure 4. This frees our computing resources (operating systems and the application software) from specific hardware platforms. A VM is migratable to new or different hardware as easily as files are copied over a network. The benefits of this are discussed later.

We are also using VM's to create a flexible for solution for GS users to run application specific code at the ground station. We enable users to upload a VM (containing their application specific software). The station simply knows how to execute a VM, and is ignorant of all application specific knowledge. VM's provide an excellent security sandbox and are resource limited (CPU and network) to prevent malicious or erroneous execution from adversely affecting GS systems. VM's are selected during contact session scheduling and can access all network accessible systems in the GS (if given proper permission). This enables them to perform custom antenna tracking, application specific data demultiplexing, and any other application process runnable from a VM. The GS is now free of application specific knowledge but still enables users to run application specific services.

7. COST BENEFIT ANALYSIS

The benefits of ROC principles as applied to terrestrial systems on the total cost of ownership (TCO) have been discussed where the cost of down time ranges from \$14K/hour for ATM services to \$2.6M/hour for credit card authorization services[28, 27, 26]. On a comparable space network system, the cost of TDRSS down time ranges from \$0.54K per hour up to \$11K per hour[4]. Since ROC attempts to reduce recovery time, this will have a positive affect on the cost of downtime.

Coding and integration time have been reduced with the use of the GSML API. New hardware components require only GSML software drivers to be written. Any changes to low level GSML specifics are masked by the autogenerated communication modules that mask GSML specifics from application writers. This simplifies the training time of new developers as well as the effects of changing the underlying communication structures. These time reductions result in a lowering of cost.

The Mercury system is an open source project with developers from universities around the world. The cost of code development and maintenance is shared across multiple universities. We benefit from the diverse background and experiences of multiple developers that lends strength and depth to the Mercury architecture. Most non Mercury specific code is also open-source (such as Linux, Apache, MySQL, and PHP). The only proprietary software we purchase is the virtual machine system from VMware.

The use of virtual machines has excellent possibilities in reducing TCO. To a host operating system (OS), a guest OS looks like several large files (usually multiple gigabytes). Migrating a VM is as simple as copying these files, which takes no more than several minutes on modern local area networks. Backups and restorations of entire systems are simple and straight forward. They can be run as hot, warm, and cold spares to facilitate rapid recovery of failed software systems. Recovery then is faster and the cost total cost of downtime reduced.

VM's also facilitate computing infrastructure upgrades. VM's can be migrated to new host systems without all the reinstallation issues of installing a system from scratch. Hardware upgrades consist of installing the hardware, installing the virtual machine monitor (which is significantly less work then a full application system install), and copying over the virtual machine.

We also use VM's to distribute our Mercury system software. In addition to the traditional source code and installation documents, we have entire VM's that can be downloaded that contain our Mercury system already installed. The use of VM's has reduced the install time from hours and days for experts and novices, respectively, to less than an hour for any user.

8. RELATED WORK

Others have noted space mission trends and are working on flexible, network-centric ground station systems. The work by the Consultative Committee for Space Data Systems (CCSDS) is an international efforts to standardize space systems at all levels. Concerning ground stations specifically, they have developed the Space Link Extension (SLE) [2]. SLE is comparable to the session level but is CCSDS specific. It doesn't provide generic data processing services, only CCSDS protocols. SLE should be extended to broaden the lower level interfaces in order to promote development of space applications (such as IP-based services). CORBA interfaces have been described to mask device heterogeneity[6]. This is an implementation of the virtual hardware level.

9. CONCLUSIONS

We have discussed the application of the end-to-end principle to ground station systems. It states that lower layers of systems should support the widest possible variety of services and functions. We have applied this to ground stations to derive a core set of services that are used across multiple missions while separating out mission specific services. We have developed a hierarchical model that describes these services and layers them based on autonomy levels.

Given this model, we briefly described our reference architecture and open source implementation, called Mercury. We are currently deploying it at universities around the world to support university satellite research missions. We are expecting to support 10-20 satellite missions launched each year with the first six launching in the summer of 2003. We have also described our use of virtual machines in the ground station. They are a primary tool in enabling flexible application level support of a ground station without it knowing any application level semantics. These systems are lowering the barriers to access space and providing basic building blocks for new operational paradigms to be explored.

At the previous RCSGSO conference in 2001, R. Holdaway asked, ”What might be done to expedite the realization of reducing the costs of ground systems[19]?” We feel the community is on the verge of a rich time of innovation. Given the explosion of Internet technologies and their application to space, this growing involvement of universities (who have significant financial constraints) may spark innovation and development to drastically increase the connectivity to space and provide ground systems at reduced costs.

10. REFERENCES

[1] VMware. `http://www.vmware.com`.

[2] Space link extension executive summary, September 2001. CCSDS 910.0-Y-0.1, Draft Yellow Book.

[3] The ground station markup language (GSML). Draft at `http://mercury.sourceforge.net/gsml/`, July 2003.

[4] Space network overview, June 2003. `http://nmsp.gsfc.nasa.gov/range/rstim.pdf`.

[5] Peter M. Allan. Developing a co-operative ground station capability. In *2nd UN/IAA Workshop on Small Satellites at the Service of Developing Countries*. IAA-01-IAA.11.3.06, October 2001.

[6] S. Bernier and M. Barbeau. A virtual ground station based on distributed components for satellite communications. In *Proceedings of 15th Annual AIAA/USU Conference on Small Satellites*, Logan, Utah, August 2001.

[7] Kul Bhasin and Jeffrey L. Hayden. Space Internet architectures and technologies for NASA enterprises. In *Proceedings of IEEE Aerospace Conference*, pages 931–941, Big Sky, Montana, 2001.

[8] George Candea, James Cutler, and Armando Fox. Improving availability with recursive micro-reboots: A soft-state system case study. In *To appear in Performance Evaluation Journal*, Summer 2003.

[9] George Candea and Armando Fox. Recursive restartability: Turning the reboot sledgehammer into a scalpel. In *Proc. 8th Workshop on Hot Topics in Operating Systems*, pages 110–115, Elmau/Oberbayern, Germany, May 2001.

[10] Maj. Gen. Craig R. Cooning. Ground systems - providing effects to the warfighter. Presented at the Ground Systems Architecture Workshop (GSAW), Mar 2003.

[11] Ed Criscuolo, Keith Hogie, , and Ron Parise. Transport protocols and applications for Internet use in space. In *Proceedings of IEEE Aerospace Conference*, pages 951–962, Big Sky, Montana, 2001.

[12] James Cutler, Peder Linder, and Armando Fox. A federated ground station network. In *Proceedings of SpaceOps 2002*, Houston, TX, October 2002. AIAA.

[13] James W. Cutler and Gregory Hutchins. Opal: Smaller, simpler, luckier. In *Proc. AIAA Small Satellite Conference*, Logan, Utah, September 2000.

[14] Donald C. Elmore and Terry R. Hurd. Fault detection and fault isolation in the ground station. In *Proceedings of MILCOM, Volume 3*, November 1997.

[15] Armando Fox and David Patterson. When does fast recovery trump high reliability? In *Proceedings of the 2nd Workshop on Evaluating and Architecting System Dependability (EASY)*, San Jose, CA, October 2002. IEEE Computer Society.

[16] Armando Fox and David Patterson. Computer Heal Thyself. *Scientific American*, pages 54–61, June 2003.

[17] Alan Freier, Philip Karlton, and Paul C. Kocher. SSL version 3.0, March 1996. Internet Draft, available at `http://home.netscape.com/eng/ssl3/ssl-toc.html`.

[18] J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext transfer protocol – HTTP/1.1. Internet RFC 2616, June 1999.

[19] R. Holdaway. How we got to be at JHU/APL: Reducing the costs of spacecraft ground stations and operations. In *Proceedings of the Fourth International Symposium on Reducing the Cost of Spacecraft Ground Systems and Operations*, Laurel, MD, 2001.

[20] David S. Isenberg. The dawn of the stupid network. In *ACM Networker 2.1*, pages 24–31, February/March 1998.

[21] David Israel. NASA TDRSS S-Band IP service developments. Presented at the Space Internet Workshop, June 2003.

[22] David Israel. STS-107 mission end-to-end IP space communication results. Presented at the Space Internet Workshop, June 2003.

[23] Mathew Long, Allen Lorenz, Greg Rodgers, Eric Tapio, Glenn Tran, Keoki Jackson, Robert Twiggs, and Thomas Bleier. A cubesat derived design for a unique academic research mission in earthquake signature detection. In *Proc. AIAA Small Satellite Conference*, Logan, Utah, 2002.

[24] Will Marchant. Status of the CHIPS mission. Presented at the Space Internet Workshop, June 2003.

[25] D. Parnas. On the criteria for decomposing systems into modules. In *Proceedings of the 1971 IFIP Congress*, North Holland, 1971.

[26] David Patterson. Availability and maintainability greater than performance: New focus for a new century. USENIX Conference on File and Storage Technologies (FAST '02), January 2002. Keynote Address.

[27] David Patterson, Aaron Brown, Pete Broadwell, George Candea, Mike Chen, James Cutler, Patricia Enriquez, Armando Fox, Emre Kiciman, Matthew Merzbacher, David Oppenheimer, Naveen Sastry, William Tetzlaff, and Noah Treuhaft. Recovery oriented computing (ROC): Motivation, definition, techniques, and case studies. Technical Report UCB/CSD-02-1175, UC Berkeley, Berkeley, CA, March 2002.

[28] David A. Patterson. A simple way to estimate the cost of downtime. In *Proceedings of LISA '02: Sixteenth Systems Administration Conference*, pages 185–188, Berkeley, CA, 2002.

[29] David A. Patterson and John L. Hennesey. *Computer Architecture A Quantitative Approach*. Morgan Kaufmann Publishers, Inc., San Francisco, California, 2nd edition, 1996.

[30] Shankar R. Ponnekanti, Brad Johanson, Emre Kiciman, and Armando Fox. Portability, extensibility and robustness in iROS. In *Proceedings of IEEE International Conference on Pervasive Computing and Communications (Percom 2003)*, Dallas, TX, March 2003.

[31] D. Reed, J. Saltzer, and D. Clark. Active networking and end-to-end arguments. *IEEE Network*, 12(3):66–71, May/June 1998.

[32] J.H. Saltzer, D.P. Reed, and D.D. Clark. End-to-end arguments in system design. *ACM Transactions on Computer Systems*, 2(4):277–288, November 1984.

[33] Trevor Sorensen, Dean Bakeris, and Richard Hieb. The design of a commercial spacecraft control network. In *Proceedings of Space Ops*, Tokyo, Japan, June 1998.

[34] Michael A. Swartwout and Robert J. Twiggs. SAPPHIRE - Stanford's first amateur satellite. In *Proceedings of the 1998 AMSAT-NA Symposium*, Vicksberg, MI, October 1998.

[35] W3C. SOAP specification. `http://www.w3.org/TR/SOAP/`.

[36] Mark Weiser. Some computer science issues in ubiquitous computing. *Communications of the ACM*, 36(7):75–85, July 1993.

[37] James R. Wertz and Wiley J. Larson, editors. *Space Mission Analysis and Design*. Microcosm Press, El Segundo, California, 3rd edition, 1999.

[38] David Zillig and Ted Benjamin. Advanced ground station architecture. In *Proceedings of SpaceOps*, Greenbelt, Maryland, November 1994.