Remote Observing from the Bottom Up:
The Architecture of the WIYN Telescope Control System.

Jeffrey W. Percival
Space Astronomy Laboratory, University of Wisconsin
1150 University Ave, Madison, WI 53706

ABSTRACT

Remote Observing has many definitions, ranging from unattended batch-mode use through simple remote logins to fully-faithful off-site observing centers indistinguishable from the on-site telescope control room. There are problems with each of these ideas: batch mode operation, for example, precludes remote interactive target acquisition and remote access to targets of opportunity. Simple remote login suffers from network problems such as full-duplex character latency; shipping screens instead of the underlying data can cause bandwidth problems and interferes with analyzing or archiving data. Brute-force reproduction of the control room requires expensive fiber or satellite connections.

The WIYN Telescope Control System was designed to be inexpensive to build and inexpensive to maintain. We emphasized the use of standard tools, portable implementations, and network friendliness. These techniques and features are precisely those that underlie a powerful Remote Observing capability. The WIYN Telescope Control System therefore supports Remote Observing from the very lowest levels, and does so effectively and inexpensively using a carefully planned architecture, standard software and network tools, and innovative methods to ship large digital images over low bandwidth connections such as phone lines. Even before the construction was complete, these techniques proved their value by allowing remote access for the purposes of eavesdropping, troubleshooting, and servo tuning.

This paper presents a block diagram and detailed descriptions of the WIYN Telescope Control System architecture. Each aspect of the Control System is discussed with respect to its contribution to the overall goal of Remote Observing, including multi-user access, bandwidth conservation, interoperability, and portability.

1 DISTRIBUTED ARCHITECTURE

The WIYN Telescope Control System is a highly distributed system. The Control System block diagram is shown in Fig. 1. The centerpiece of the system is the message hub, which we call the message *router*. All other processes are clients of the router, including the TCS process (in this paper, the acronym TCS will refer to the software process actually controlling the telescope; "Control System" will refer to the system as a whole, shown in Fig. 1). Each client connects to the router via TCP sockets, and sends a subscription list telling the router what kind of message traffic it is interested in receiving. All WIYN messages are labeled with a four-part identifier, naming the system, subsystem, device, and attribute of the data contained in the message. For example, the client might move the telescope with a (WIYN,TCS,AZIMUTH,SET) message and look for its position by subscribing to a (WIYN,TCS,AZIMUTH,STATUS) message.

Each function in the Control System was, where possible, allocated to a separate, single-purpose process. For example, the Command Line Interface (CLI) receives no data from the Control System (it is a write-only client) and our simple, character-based display program writes no data. The message archiver process simply reads messages and writes them to disk. Of course there is a higher-level Graphical User Interface (GUI) that combines commanding and data display, but its connection to the router is no different than any other.

2 MULTI-USER ACCESS

We designed our Control System to support multiple users simultaneously. Scientists, engineers, programmers, and students can connect to and disconnect from the system at will (subject to satisfying the access control requirements) with essentially no effect on the TCS itself. Note that the TCS is *not* the object of such connections, so that its load does not change as connections come and go. The TCS is aware only of the router, and sends data only to it. The router handles data subscriptions in a fashion similar to e-mail exploders.

2.1 Asynchronous control

We chose an asynchronous, message-oriented model over a synchronous one based on remote procedure calls (RPCs). The latency problems associated with networks and the desire to provide sequencing, arbitration, and verification seemed to make an asynchronous model more appropriate. Users send messages into the router where they are validated and sent on to subscribers. Message buffering exists at each end of the socket connection, as well as in the underlying network transport protocol (TCP), so degraded network performance most often results only in a little burstiness in response time.

2.2 Serial device servers

We strove to bring our distributed, networked, multi-user approach even to parts of the system that normally do not have these characteristics. The WIYN Control System uses a number of serial devices, including airport-style status displays, handpaddles, dome motor inverters, uninterruptible power supplies, and a weather station. We designed "serial device servers" for each of these devices. The device server manages the serial line to the device and accepts socket connections from the network. It collects and sequences device-bound traffic from the socket connections, and explodes device data to the currently connected clients.

This has turned out to be a *very* useful capability. The TCS is just one of many possible clients of each serial device. An operator, for example, can run a process on an observatory workstation that connects to the dome encoder server and provides a simple time-ordered list of encoder values. This can be done while the TCS is running, without interfering with telescope operations, or it can be done when the TCS is down, thereby relieving the operator of having to start up the whole system merely to acquire the output of a serial device.

2.3 Stateless pointing kernel

It is possible, within the constraints of the security and safety measures imposed, for more than one user to control the telescope. This brings up some obvious problems associated with the presence of multiple, and possibly conflicting, telescope commands. We chose to finesse this problem by designing the TCS to meet the following requirements:

1. do no harm to the hardware
2. make a best effort to execute commands
3. execute the most recently received command

Item 1 merely means that the TCS knows where the hard stops are, and if commanded to go past them will choose instead to come to a soft, no-overshoot stop just short of the limit switches.

Item 2 means that the TCS does not do a lot of command validation and error checking, other than what is necessary to enforce Item 1. For example, if a target is specified that is below the horizon, the TCS will send the telescope down until the Item 1 software takes effect and smoothly stops the telescope. As the sky rotates below the horizon, the telescope will glide along the lower soft limit, and should the target rise, the telescope would pick it up at the soft limit and track it upwards as expected.

Item 3 relates to how the telescope deals with sequential but conflicting pointing commands. We decided to eliminate completely the idea of slewing and trajectory management. This simplifies the TCS design in that there is no trajectory set-up time, and no explicit trajectory mode that the telescope can be in the midst of when a new pointing command arrives. The TCS maintains two models of the telescope mount. One is the "virtual" telescope, which has zero mass, moves instantaneously and discontinuously, and can point anywhere on a sphere, even below the horizon. The real telescope is represented by the current 5-axis encoder readings (azimuth, elevation, dome, and two Nasmyth instrument rotators). When a pointing command arrives, the virtual telescope moves instantaneously to the new point in the 5-axis coordinate space. The real telescope moves slowly, so there is initially a large 5-axis error. Each axis is separately subjected to a simple discontinuity smoother, based on the idea that for a maximum deceleration to the origin of a coordinate space, one should follow a trajectory in a position/velocity phase space whose velocity is proportional to the square root of the error. This is stateless in that each pointing command simply represents a translation of phase space axes for each telescope axis. The smoothing algorithm simply keeps trying to reach the square-root geodesic, and having reached it, drives the axis to the origin where both the axis position and velocity *simultaneously* reach their desired values. There is no overshoot.

This statelessness relates to Remote Operations by moving the states out into the clients, thereby removing state conflicts from the kernel. For example, the GUI is capable of deciding that a target is below the horizon, and can therefore simply decide not to send that command. Multiple users can coordinate their pointings before the TCS sees the commands, but if conflicting commands should arrive in sequence, the TCS will simply obey as best it can rather than, say, abort a trajectory and respond with error messages. We recognize the need for a coherent collaborative use of the telescope, but we feel that building states into the TCS is not the way to resolve usage conflicts.

3 BANDWIDTH CONSERVATION

One of the most critical resources for remote operations is the allocation and use of network bandwidth. Astronomical images are large and getting larger, but the remote location of most observatories means that getting high bandwidth connections to the site will remain an expensive option for some time to come. One way of providing a remote operations capability is to create one or more remote operations centers, each of which is connected to the primary site by dedicated high bandwidth network connections. Observers might go to the remote operations center rather than to the observatory itself.

We defined the remote observing problem in a different sense. Rather than make the bandwidth fit the data, we decided to see how much we could make the data fit the bandwidth. For a worst case analysis, we chose to design the system to be compatible with a 28.8 k bits per second (kbps) modem connection. This worst case uses simple, inexpensive hardware that is commonly available to everyone. The obvious advantage over a dedicated remote operations site is that by staying within the phone system, the observer can bring observing onto the desktop or into the home.

There are three components to the WIYN remote observing capability. The TCS generates engineering data

(e.g. voltages, encoder readings, software settings), the instruments generate science data (e.g. 1024x1024 CCD images) and the observatory video system generates TV images (e.g. guide stars, dome cameras, cloud monitors).

3.1 Engineering data

The WIYN Engineering Data Subsystem (EDS) is similar to spacecraft telemetry systems. Every second, a frame of data (in the form of a vector of integers) is sent out from the TCS to the message router. This vector is sent to each Control System client. This EDS frame is about 2800 bytes long, resulting in a data rate of about 22 kbps. We do a simple compression to reduce this bandwidth. Within any 60-second interval, only difference frames are sent (the difference between the current frame and the baseline frame acquired at the previous 60-second time point). These difference frames compress well using simple run-length encoding, and result in an effective bandwidth of only about 4 kbps. Once per minute, an uncompressed full frame is sent so that new clients can synchronize themselves to the difference frames.

3.2 Image data

This is the toughest part of the problem. Lossless transfer of large digital images will consume almost any available bandwidth, so we confined the problem to the most pressing needs of the observer: target acquisition and quick-look data reduction. For each of these needs, it is almost never necessary to send *all* of the bits, as long as all of the *useful* bits can be sent. The crux of the compression problem is therefore to identify the useful bits. (Astronomical images are often noisy, which precludes any form of lossless compression.) Throwing away the faint bits doesn't work well, as astronomers are often looking for things near the background level. A common technique of lossy compression is to transform the image into some other data space, such as a Fourier space, and discard data there. The problem with Fourier bases, which underlie the Discrete Cosine Transform of some compression schemes, is that discarding low power coefficients can introduce undesirable artifacts in images containing point sources and sharp edges.

White[1] has developed a wavelet-based compression scheme that performs exceptionally well for lossy compression of astronomical images. The 2-D Haar transform[2] is a fast, exactly reversible integer transform that performs a multiresolution decomposition of an image. The transformed image lends itself to progressive transmission in that the receiving side can invert the transform when as little as 1% of the total amount of data have been received, resulting in a lossy reconstruction that is visually very similar to the original. At compression factors of 10, the lossy reconstruction is often visually *indistinguishable* from the original. Fig 2 shows a normalized RMS error Q, as a function of fraction of data received (lossiness), for three methods of progressive transmission using a typical astronomical image.

The raster method sends the raw image row by row, like a FAX machine. Convergence to the original image is nearly non-existent until almost all ($> 95\%$) of the image has been sent. This method is obviously unsuitable for progressive transmission because the lossiness is zero for the part of the image already received, but 100% for the part not yet received. Sending the raw image in bitplane order converges more quickly, but the image suffers from intensity quantization and brightness thresholding during much of the transmission. Sending the 2-D Haar transform in bitplane order converges much more quickly, and therefore at much higher lossiness, than the other methods.

We have incorporated this compression and progressive transmission into a prototype system[3,4] for the WIYN Observatory. We have sent an 800x800 pixel 16-bit image over a 2400 baud connection, and received in less than 60 seconds a version of the image that did not differ appreciably from the original. We note that sending this image uncompressed over the same line would have taken 71 minutes. Another desirable feature of the system as implemented is that the degree of lossiness is not built in at the observatory end. A user can decrease the lossiness

simply by waiting longer, ultimately resulting in a completely lossness transfer if the full 71 minutes is allowed to pass. Users whose needs are satisfied quickly, say for simple target identification, can abort the transmission and move on after only a tiny fraction of data have been sent.

With the promising results of the 2400 baud test, we plan to allocate about 10 kbps for digital image transmission.

3.3 Remote video

The remote video capability is an offshoot, still under development, of the wavelet-based progressive image transmission described above. Video frames are captured by a Silicon Graphics Indy workstation, which adds successive frames using a leaky memory algorithm. The integrated video image is passed off to the progressive image transmission server as often as the allocated remote video bandwidth will allow, to be sent on to the client.

We plan to allocate 10 kbps for remote video. With the good lossy performance of the progressive image transmission described above, we expect to be able to send about 2 to 4 frames per minute with this technique.

4 INTEROPERABILITY

Interoperability is imperative in the WIYN distributed environment. All of the data going into and out of the TCS are put into network byte order, and only integers are put into messages. Floating point numbers are broken into two parts, an integer portion and a scaled fractional portion, before encapsulation. The codec routines are shared between the TCS and the clients, so that no format compatibility problems arise.

Images are stored and sent using the FITS astronomical image format[5], which provides a machine-independent method of encoding an image. The encoding information is embedded in the image file, allowing any conforming FITS reader to access the data.

5 PORTABILITY

We wanted portability not only in the user interfaces to the WIYN Control System, but in the TCS software itself. This would allow us to be able to port the TCS easily to other real-time environments, as well as to run it on non-real time workstations as a simulator. Having a high fidelity simulation of the actual TCS is very desirable for system development, debugging, and for off-line work such as verification of observing sequences and user training. We felt that the only way to keep the simulation accurate and faithful was to run the TCS *itself* on a workstation, rather than rewrite a second, off-line version of it.

To achieve this, we chose Unix workstations as the non-real time components and a Unix-like real-time operating system, VxWorks, to run the real-time hardware. In addition, we made extremely parsimonious use of VxWorks-specific code. Some use of code specific to the real-time environment is unavoidable: semaphores, for example, to sequence tasks in a multi-tasking environment and to ensure coherent use of shared resources. We isolated such code to a few modules, and even in those modules used compiler preprocessor directives (`ifdef`, `endif`) to allow the module to be used either as a standalone task in VxWorks or as a simple procedure in a compiled and linked workstation program.

We also resisted the temptation to use "convenience" code supplied by the vendor. For example, the operating

system comes with a very nice ring buffer library. We chose to produce our own such routines, where needed, to avoid *requiring* the vendor's environment when running the TCS.

These choices worked very well for us. The TCS code runs in real-time on a Heurikon V3D, but we can run the same code, right down to the inner servo loops, on DECstation 5000's running Ultrix, Ultrix Vaxes, Sun Sparcstations, SGI Indys and Indigos, and DEC Alpha (OSF/1) machines.

6 CONCLUSIONS

We have taken the position that a modern telescope control system can be distributed, portable, and machine-independent, with its principle operations (including quick-look science data) able to be carried out using communications bandwidths available over conventional phone lines. The WIYN Telescope is still being commissioned, so the design described in this paper has yet to be put to the real test, but our experience with the Control System so far is promising. We routinely connect to the observatory for engineering and software tests, and have used the progressive image transmission system to send images to remote sites. Our engineers found the Engineering Data System so useful that they could test and tune the servo loops remotely. We are optimistic that these design features will prove useful for remote observing when WIYN science operations begin.

REFERENCES

1. White, R. L. "High Performance Compression of Astronomical Images", 1992, available via ftp at stsci.edu.

2. Haar, A. Math. Ann. 69, 331, 1910

3. Percival, J. W. and White, R. L. "Efficient Transfer of Astronomical Images Over Networks" in Astronomical Data Analysis Software and Systems II, eds. R. J. Hanisch et al., (San Francisco: Astronomical Society of the Pacific).

4. White, R. L. and Percival, J. W. "Compression and Progressive Transmission of Astronomical Images" SPIE Technical Conference 2199, 1994

5. Wells, D. C., Greisen, E. W., and Harten, R. H. "FITS: A Flexible Image Transport System" Astronomy and Astrophysics, 44, 363-370, 1981.

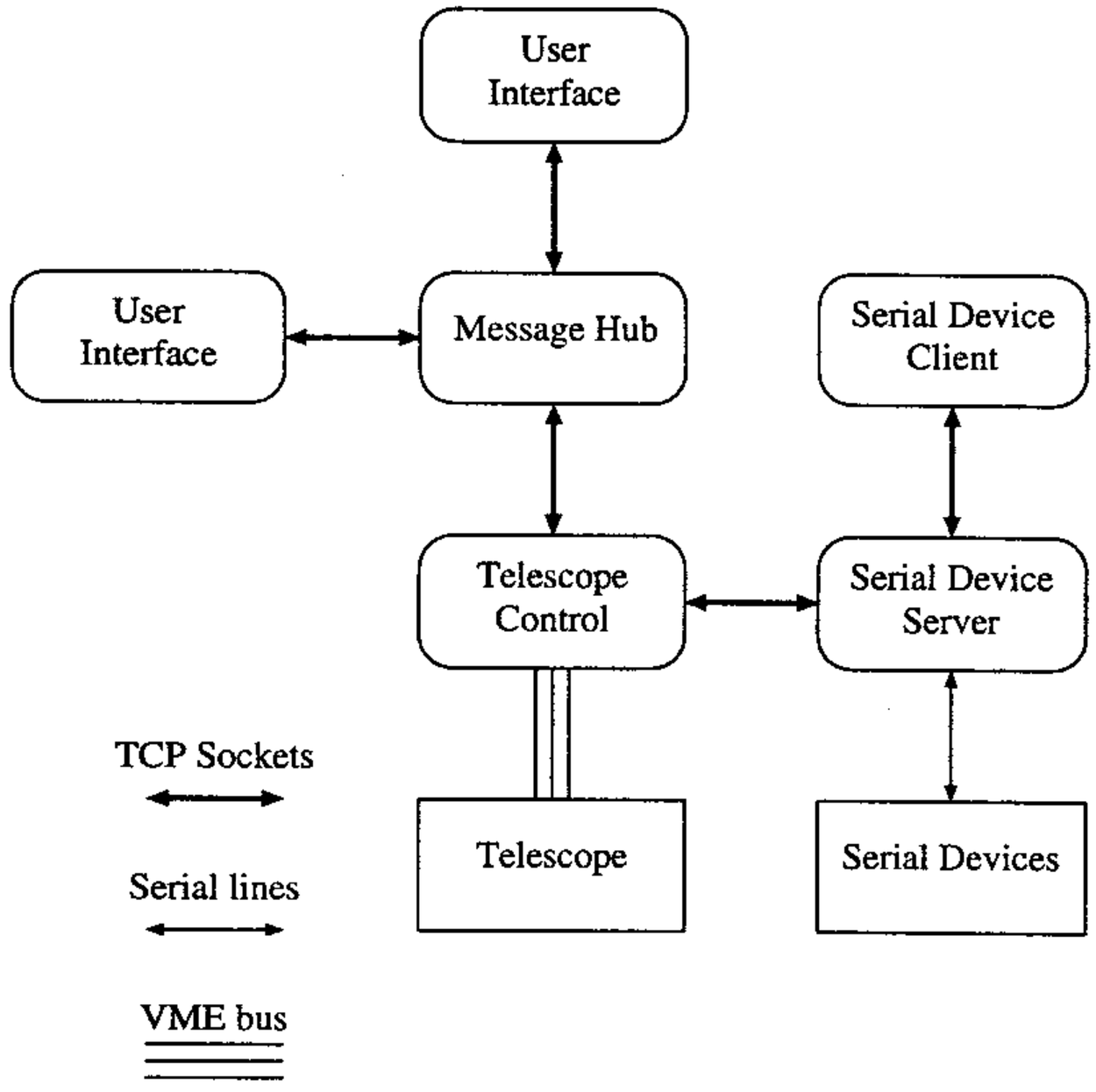

Figure 1: WIYN Control System Architecture

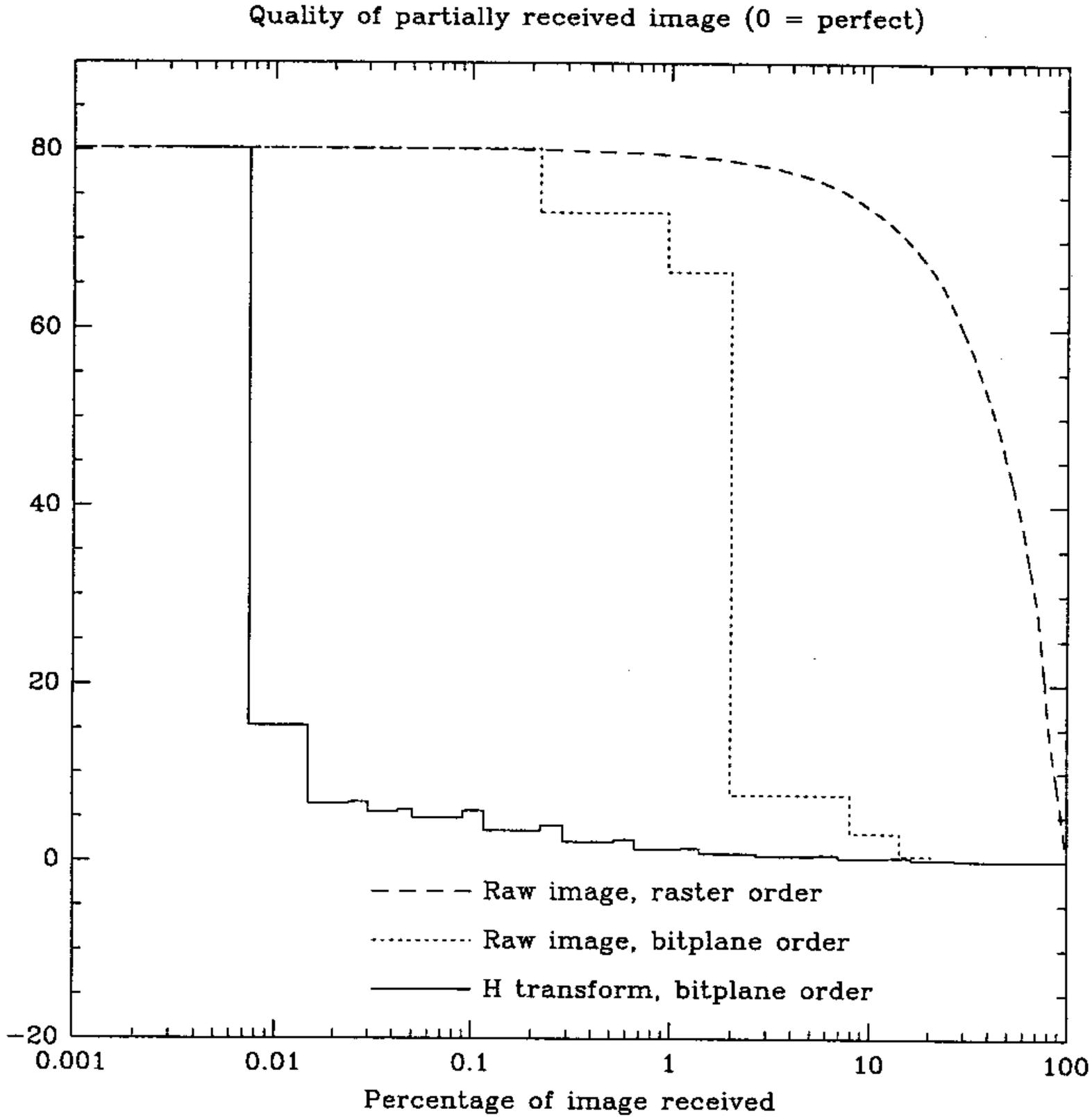

Figure 2: Progressive Image Transmission Quality