Network Working Group
Request for Comments: 928

M. A. Padlipsky
Mitre Corp.
December 1984

INTRODUCTION TO PROPOSED DOD STANDARD H-FP

Status Of This Memo

This RFC suggests a proposed protocol for the ARPA-Internet community, and requests discussion and suggestions for improvements. Distribution of this memo is unlimited.

Important Prefatory Note

The broad outline of the Host-Front End Protocol introduced here and described in RFC 929 is the result of the deliberations of a number of experienced H-FP designers, who sat as a committee of the DoD Protocol Standards Technical Panel under the author's chairmanship. The particular protocol to be described is, however, the result of the deliberations of a small, ad hoc group, who sat as a de facto subcommittee of the H-FP committee, also under the author's chairmanship. The protocol, then, follows the consensus of the full group as to what the new H-FP should "look like," but has not benefitted from painstaking study by a large number of experienced H-FP designers and implementers. (It has been looked at before release as an RFC by several of them, though.) Even if that were not the case, it would still be the intent of the designers that the protocol be subjected to multiple test implementations and probable iteration before being agreed upon as any sort of "standard". Therefore, the first order of business is to declare that THIS IS A PROPOSAL, NOT A FINAL STANDARD, and the second order of business is to request that any readers of these documents who are able to do test implementations (a) do so and (b) coordinate their efforts with the author (617-271-2978 or Padlipsky@USC-ISI.ARPA.).

Historical/Philosophical Context

Late in May of 1971, the author was presenting a status report on whether the Multics ARPANET implementation would be ready by the July 1 deadline declared by the sponsor earlier that month. Some controversy developed over the fact that the Multics "NCP" (Network Control Program--actually a blanket term covering the Host-Host and Host-IMP protocol interpreters) did not queue requests for connections. As the specification explicitly declared the topic to be one of implementors' choice, the author attempted to avoid the argument by asking the interrogator what he was up to these days. The answer was, "Oh, I'm working on the High-Speed Modular IMP now" (later the Pluribus IMP). And the proverbial coin dropped: The author replied, "I've got a great idea. Now that we've got some space to program in the IMP, why don't we separate out most of the

NCP and do it outboard: the only thing that really matters in the Host is associating sockets with processes, and if we had common implementations of all the bit-diddling stuff in the IMPs, we wouldn't have disputes over the interpretation of the spec and we'd also save a lot of Host CPU cycles!"

As far as the author knows, that incident was the beginning of what came to be called "Network Front-Ends" and, more recently, "Outboard Processing Environments." (The name change, by the way, was motivated by a desire to prevent further confusion between NETWORK Front Ends--always conceived of as distributed processing mechanisms for the offloading of intercomputer networking protocols from Hosts--and traditional communications front-ends, which have no connotation of bearing protocol interpreters invokable by Host-side programs.) At least, the idea was original to him and he later was a principal designer and the primary author of the first Host-Front End Protocol. So, on the one hand, the present document might be marred for some readers by undertones of parental pride, but on the other hand, if you like primary sources....

The evolution of the outboard processing idea has been dealt with elsewhere [1]. For present purposes, it should suffice to observe that some half-a-dozen implementors of "NFE's" of various sorts are known to the author to have met with success. The topic of why use an explicit protocol in the first place (as opposed to emulating a device, or devices, already known to the Host/operating system) deserves a word or two here, however. ([2] deals with it in more general terms.) The crucial consideration is that in the general case you wind up "not doing real networking" if you attach a Host to a network by known device emulation, where real networking is taken to mean what has been called "resource sharing" in the ARPANET literature, and what appears to be dubbed "open system interconnection" in the ISO literature: Operating systems' built-in assumptions about known devices--whether terminals, terminal controllers, or RJE stations--tend to get in the way of the sort of process-process and eventually procedure-procedure communications that serve as the basis for applications more interesting than simple remote login. To those unfamiliar with the outboard processing approach, the premise that the way to attach is via an explicit protocol may be difficult to accept, but to those who have done it, it makes almost perfect sense.

To those, by the way, who have worked in intercomputer networking from the perspective of inboard (Host-side) implementations of protocol suites, the outboard processing idea often seems to lead to less than optimal results, especially as to maximizing throughput. And it is difficult to argue that if a given Host were well and truly

fine-tuned to "do networking" the insertion of an extra processor could somehow lead to better networking. However, for Hosts where conservation of CPU cycles is an issue, or even where memory is scarce (i.e., where it's desirable to conserve the resources being shared), outboarding is clearly the way to go. For that matter, viewing outboard processing aright (as a form of distributed processing) it can be argued that even for extremely powerful "intelligent work stations"/"personal computers" which have the resources to spare it still makes sense to outboard in order not to have to do new implementations of entire protocol suites for each new such system--always assuming, of course, that the Host-Front End protocol in play is noticeably less complex than the offloaded protocols.

None of this is meant to imply that outboard processing is the ONLY way to do intercomputer networking, of course. It is, however, meant to suggest that outboard processing can be advantageous in a number of contexts. Indeed, given the joint advents of microprocessors and Local Area Networks, a generic bus interface unit which also plays the role of a NFE (that is, is an Outboard Processing Environment) even allows for the original intent of "offloading to the IMP" to be realized, so that a free-standing, possibly fairly expensive NFE need not be interposed between Host and net. Note, by the way, that nothing in the OPE approach requires that ALL Hosts employ OPEs. That is, the only protocols "seen" beyond the Comm Subnet Processor are the common intercomputer networking protocols (e.g., all DDN IMPs see and read IP datagrams). H-FP is strictly a matter between a Host and its OPE.

It is also important to be aware that, given the advent of several different suites of protocols in the networking world, it might well be the case that the only reasonable way to achieve "interoperability" might well be to use a suitable H-FP (such as the one to be presented in the companion RFC) and an Outboard Processing Environment which is capable of parallel invocation of protcol suites (with the choice of suite for a given connection being dependent, of course, on the native suite of the desired target Host and/or application).

The unquestionable advantages, then, of the approach, based on ten or more years of experience and analysis, would seem to be as follows--always recalling the assumption that the work to implement and execute the H-FP in play is small compared to the full protocol suite in question: As noted, common implementation of a protocol suite has the automatic advantage of mutual consistency; further, particularly in the DOD context, it's far easier to procure common

implementations of standard protocols than to procure different ones on a per-Host type basis. Also as noted, if the resources to be shared are viewed as being the participating Hosts'

CPU cycles and memories, these resources are conserved by doing as much as possible of the networking protocols in an OPE rather than in the mainframe. Another, less evident advantage is that having an OPE effectively insulates a Host against changes in the outboarded/offloaded protocols--or even changes of the protocols, should the nascent international protocol standards ever mature sufficiently to supplant the in-place DOD standards. (That is, given an abstract enough interface--in the spirit of the Principle of Layering--a Host could, for example, go from doing TCP as its "Host-Host" protocol to, say, ECMA Class 4 as its "Transport" protocol without taking any particular cognizance of the change, however unattractive such a change would be to advocates of the APRANET Reference Model such as the author. See [3] for more on the implied "Reference Model" issues.) Finally, although a few rather specialized points could also be adduced, it should be noted that for network security architectures which are predicated on the ability to control all means of egress from and ingress to "the net", uniform use of OPEs is clearly desirable.

If we can stipulate that an OPE is/can be a good thing, then the remaining problem is just what the protocol interpreted by a Host and its OPE ought to be, once it's observed that a standard protocol is desirable in order to allow for as much commonality as possible among Host-side interpreters of the protocol. That is, we envision the evolution of paradigmatic H-FP PIs which can more or less straightforwardly be integrated with various operating systems, on the one hand, and the ability simply to transplant an H-FP PI from one instance of a given operating system to other instances of the same system, much as is currently being attempted in the DODIIS NFE program. Again, the major motivation in the DOD context is the minimizing of procurement problems.

Technical Context

As noted, some half-a-dozen Host-Front End protocols have been seen by the author. Indeed, in December of 1982, a meeting was convened to allow the developers of those H-FPs to compare their experiences, with an eye to coming up with a proposal for a DOD standard H-FP; this paper is a direct result of that meeting. In the current section, we present the consensus of the meeting as to the broad outline of the protocol; in the accompanying document, the current version of the proposed protocol will be presented, as detailed by the author and Richard Mandell and Joel Lilienkamp (both of SDC).

Note, by the way, that in some sense we should probably have changed the name from H-FP to H-OPEP (or something), but the habit of saying "H-FP" seems too deeply engrained, despite the fact that it does seem worthwhile to stop saying "NFE" and start saying "OPE." (Besides, H-OPEP looks rather silly.)

A final preliminary: all the designers and implementors of H-FPs present at the December meeting concurred that the true test of any protocol is how well it implements. Therefore, until several implementations of the "new" protocol have been performed and assessed, it must be understood that the proposed protocol is precisely that: a proposal, not a standard.

Not too surprisingly, the first point on which consensus was reached is that there are three separable aspects (or "layers") to an H-FP: At bottom, there must be some physical means for conveying bits from Host to OPE and from OPE to Host. As it has always been a premise of outboard processing that the Host's convenience is paramount, just what this physical layer is can vary: typically, a bit-serial interface is customary, but parallel/DMA interfaces, if available for the Host and interfaceable to a given OPE, are fair game. (So would teleporting the bits be, for that matter.)

In the middle, there must be a layer to manage the multiplexing of network "connections" and the control of the flow between Host and OPE. If we agree to call the lowest layer the Link and the middle layer the Channel, one thing which must be noted is that between the two of them, the Link and Channel layers must be responsible for reliably conveying the bits between Host and OPE. After all, an OPE'd Host should not be "weaker" than one with an inboard implementation of a robust Host-Host protocol such as TCP. It should be noted that any Host which "comes with" a suitable implementation of the X.25 interface protocol (where the definition of "suitable" is rather too complex to deal with here) could, given an OPE conditioned to accept it, quite cheerfully satisfy the requirements of the lower two layers. This is not to say that X.25 "is" the mechanization of H-FP's Link and Channel layers, however; merely that it could be used. The protocol spec itself will detail an alternative, less cumbersome channel layer for Hosts which don't have or want X.25.

The top layer of H-FP is the most important: we refer to it as the Command layer. Here is where the peer H-FP modules in a given Host and OPE communicate with each other. Indeed, the segregation of JUST multiplexing and flow control (plus reliability) into the Channel Layer is done--in addition to making it easier for Hosts that possess preexisting software/hardware which could be turned to the purpose--so as to clarify "what the H-FP is": it's the commands and

responses of the Command layer wherewith the Host's processes are able to manipulate the outboard implementations of the members of a protocol suite. The use of the phrase "commands and responses" is rather significant, as it happens. For in the protocol to be proposed for DOD standardization, unlike all but one of its predecessors, binary encoded "headers" are not employed; rather, the H-FP commands are indeed ASCII strings, and the responses (following the practice of ARPANET FTP) ASCII-encoded numbers.

There are various reasons for this departure, which initially stemmed from a desire to have the same NFE be usable for terminal traffic as well as Host offloading, but the one that seemed to dominate when consensus was arrived on it as the basis for the new standard is that it is very much in the original spirit of H-FP. That is, if you want to "make things as easy as possible for the Host", it makes a great deal of sense to offload in a fashion that only requires some sort of scenario or script ("exec-com"/"command file"/"shell command" are approximations on some systems) in the Host, rather than requiring a program, possibly of more complexity than we would like. This is not to say that we envision all--or even most--Hosts will take the scenario approach to H-FP mechanization, but rather that the command orientation chosen allows for the possibility. (It would be useful to recall that the Channel layer does all the necessary multiplexing/demultiplexing, so that each channel's metaphorical state machine--at least on the Host side--really has very little to worry about other than "doing its thing.")

It should be noted that the proposed protocol provides a mechanism for offloading "all" protocols. That is, although most "first generation NFEs" only handled ARPANET Reference Model Layers II and I (Host-Host and Network Interface--approximately ISO levels 4-1, with some of L5's functionality included when it comes to service identifications being handled via Well-Known Sockets in L II), it is assumed that OPEs will be evolved to handle L III offloading as well (ISO 5-7). Indeed, it should also be noted that what is being addressed here is "the protocol", not "the" OPE. More will be said on this topic below, and in the protocol spec itself, but it is important to realize from the outset that the H-FP being proposed is intended to be implementable by any number of OPE suppliers/vendors, so "an" OPE may or may not choose to implement, say, a given file transfer protocol, but provided it says so in proper H-FP terms and does offload some other protocols it's still an OPE in our sense of the term. (Cf. "Issues" and "Non-Issues", below.)

Issues

The following items are either in some sense still open issues or bear special emphasis:

Command Approach

The most striking feature of the new H-FP, especially to those who have seen older H-FPs, is the decision to employ character-oriented commands rather than the more conventional binary-oriented headers at the Command Layer. As noted, the primary motivation was the report that the approach worked well when it was employed in an H-FP for the Platform Network called NAP (Network Access Protocol) [4]. In discussions with NAP's originator, Gerry Bailey, the author was convinced of the fundamental reasonableness of the approach, but of course that doesn't have to convince others. Additional rationales emerged in discussions with Gary Grossman, the originator of the DCA/DTI H-FP [5], which is probably the best-known current H-FP and which furnished the default Channel Layer for the new one: In the first place, the text approach makes parsing for the ends of variable-length parameters easier. In the second place, it allows for the possibility of creating a terminal-supporting OPE in a very straightforward fashion should any OPE developer elect to do so. (See below for more on the distinction between OPE developers and H-FP implementors.) Finally, there's nothing sacred about binary headers anyway, and just because the text approach is different doesn't make it "wrong". So, although it's not out of the question that the new protocol should back off from the text approach if reviewers and/or implementors come up with compelling reasons for doing so, the already frequently encountered reaction of "it feels funny" isn't compelling. (It was, indeed, the author's own initial reaction.) Besides, "nobody" (not even Gary) really liked the top layer of the DCA/DTI H-FP.

X.25 Appropriateness

Of more concern than how text "feels" is whether X.25 "works". That is, we understand that many system proprietors would greatly prefer being able to use "off-the-shelf" software and hardware to the greatest extent feasible and still be able to do intercomputer networking according to DOD Standards, which is a major reason why we decided to take the H-FP commands out of the Channel Layer of the DCA/DTI H-FP even before we decided to encode them as text. However, it is by no means clear that any old vendor supplied "X.25" will automatically be usable as a new H-FP Channel and Link layer mechanization. As noted, it all depends upon how Host

programs (the Command Layer/H-FP Protocol Interpreter in particular) are able to invoke X.25 on particular systems. Also, there might be peculiarities in the handling of some constructs (the Group and Member fields--or whatever they're called--are a strong candidate) which could militate against getting JUST demultiplexing and flow control out of X.25-as-Channel Link/Layers. For that matter, it's conceivable that on some systems only one process can "own" the presumed DCE, but there's no interprocess communication available between it and the processes that want to use H-FP. What that all amounts to, then, is that we don't pretend to be sufficiently versed in the vagaries of vendor-idiosyncratic X.25 implementations to claim more than that we THINK the new H-FP Command Layer should fit "on top of" X.25 in a Host such that a suitably crafted OPE could look like a DCE to the low-level Host software and still be an OPE in our sense of the term. Finally, some reports on bit-transfer rates attainable through typical X.25 interfaces give rise to concern as to whether such a lash-up would be "good" even if it were feasible.

DCA/DTI Channel Layer Appropriateness

The Channel Layer of the DCA/DTI H-FP has been implemented for a few Host types already, and is being implemented for others (in particular, as part of the DODIIS NFE project). A delicate decision is whether to alter the header structure (e.g.--and perhaps i.e.--to remove the now-superfluous command and response fields). On the "con" side are the considerations that implementations DO exist, and that it's well specified. On the "pro" side are that keeping the header as it is is in some sense "wasteful" and that somebody's going to have to go over the spec again anyway, to remove that which no longer applies. (It should be noted that Gary Grossman was initially tempted to scuttle the Group and Member trick, but the presence of a similar dichotomizing in X.25 seems to rule that out.) One of the interesting issues during the review phase of the new H-FP, then, will be the decision about which way to go on the Channel Layer header in its non-X.25 version. (NOBODY considers going X.25 only, be it noted.) By the time the protocol is finalized, it will, of course, be made clear in the protocol spec, but I'll probably leave this in the final version of the Introduction just for historical interest anyway.

Syntax

Another point which probably needs close scrutiny during the review process is the "syntax" of the command lines. Basically,

we just took our best shot, but without any claims that it's the best possible way to express things. So comments and/or alternatives are earnestly solicited on this one.

L III Offloading

Contrary to the expectations of some, we are allowing for the offloading of Process/Applications Layer (ARPANET Reference Model L III) protocols. Both Bailey and Grossman reported favorably on the feasibility of this. Two points should be made, however: It's perfectly fair for a GIVEN OPE implementation not to offload a given L III protocol, although it would presumably not sell as well as ones which did. That is, we're not claiming that by inventing a mechanization of the feature in the spec we levy a constraint on everybody who implements "the protocol", (Cf. Fabrication under Non-Issues, below). Just as we were feeling our way on syntax in general, we're really feeling our way when it comes to the L III stuff. (I'm not even sure I managed to convey what I meant for "mediation level" to Joel and Dick.) Again, suggestions are solicited.

Security

During the detailed design pass, we had an intensive discussion with some of the Blacker design team on the interplay between the new H-FP and a meant-to-be multilevel-secure OPE such as Blacker. The conclusion was that by and large "Security" is to be an aspect of an enhanced H-FP, rather than the standard one. The reasoning was rather involved, but seems to amount to the following: Hosts that are NOT MLS (or "Compartmented") have two significant properties in our context: They're in the vast majority of present-day systems. They have no legitimate need even to tell their OPEs what they "think" their current System High or Dedicated Mode level is; that information should be furnished by some trusted portion of a network security architecture (e.g., a security enhanced OPE, or a table in a "secure" comm subnet processor).

Thus, even having the optional security label/level field in the Begin command is in some sense overkill, because we're not sure of any sensible circumstances in which it would be useful, but we put it in "just in case". On the other hand, Hosts that ARE MLS/Compartmented by definition can be permitted to assert what the level of a given transmission (or perhaps of a given connection) should be, and their OPEs need to have a mechanism for learning this. But it is by no means clear that a given Host (or even a given OPE) will be so structured as to make the H-FP PI,

the Channel PI, and the Link PI ALL trustworthy--as they'd have to be if the security labeling were part of H-FP. So, we envision the labeling's being handled by trusted code in both Host and OPE that will be inserted into the normal processing route at the appropriate point for the given architecture (presumably "at the very bottom" of the Host, and "the very top" of the OPE), and that will place the label in a convenient, known position in the Host-OPE transmission "chunk" (block/packet/data unit) as the circumstances dictate. (It's likely--but we wouldn't swear to it--that a good place would be just before the H-FP command, and if that's the case then semi-clearly the security enhanced H-FP PIs would have to "make room" for it in the sense of handing the Channel Layer a suitably lengthened "chunk".)

The Host and its OPE should be viewed as a single entity with regard to labeling requirements in the non-MLS/C case, and either the OPE will be conditioned to emit the right label or the CSNP will "know" anyway; in the MLS/C Host and OPE case (and it should be noted that it's just about impossible to envision a MLS/C Host which IS outboarded which DOESN'T have a MLS/C OPE) it will depend on the given security architectures as to whether each "chunk" needs labeling (i.e., there COULD be trusted H-FP, Channel, and Link PIs, so that only at channel establishment time does the label need to be passed), but it seems likely each "chunk" would need labeling, and we can see how that would happen (as sketched above).

This is all, of course, subject to reappraisal when the full-time Security folks get in the act, but for now, H-FP per se is viewed as playing no direct role in "Security"--except indirectly, as noted below under the Symmetric Begins Non-Issue. (In case anybody's worrying about the case where the OPE is physically remote from its Host, by the way, that line would have to be protected anyway, so the Host/OPE-asa-single-unit view should hold up.)

How It Implements

The final issue to take note of is that one of the central premises of the Outboard Processing approach has always been that H-FPs can be invented which implement more compactly on the Host side than the code they're allowing to be offloaded. We certainly think the new H-FP will fulfill that condition, but we'd certainly like to hear of any evidence to the contrary.

Non-Issues

The following items are declared to be non-issues, in the sense that even though some people have expressed concern over them we believe that they are either "not part of the protocol" or resolved already for reasons that were overlooked by those worried about them:

Fabrication

Who builds OPEs isn't within our purview, except to the extent of hoping a few volunteers come forward to do testcase implementations of what is, at present, only a paper protocol. However, beyond agreeing that a few points should be marked as "Notes to Entrepreneurs" in the spec, we didn't attempt to dictate how an OPE vendor would behave, beyond the explicit and implicit dictates of the protocol per se. For example, if a given OPE doesn't offload SMTP, it jolly well ought to respond with the appropriate "Function not implemented" code, and if a vendor claims to accept X.25 for Channel and Link disagreements over what X.25 "is" are the province of the vendor and the customer, not of the H-FP spec. As OPE'S are supposed to be offloading COMMON protocols in a COMMON fashion, a given OPE should be able to interoperate with another Host irrespective of whether that Host even has an OPE, much less whose OPE it is if it's there. Thus, for example, even though you'd expect to find OPEs that "come with" their own LANs as a fairly frequent product, we don't appeal to the notion in the conceptual model; nor do we attempt to dictate "chunk" sizes at the Channel level. A protocol spec isn't an implementation spec.

Symmetric Begins

For almost as long as there have been H-FPs, there has been disagreement over whether only the Host can begin a connection or if the OPE can also take the initiative. I am delighted to be able to resolve this one finally: It turns out there IS a compelling reason for insisting that THE PROTOCOL include provision for OPE --> Host Begins, so it's "in" the protocol--but any Host that doesn't need to deal with them doesn't have to (just "spell" the "Function not implemented" response code correctly).

(In case anybody cares, the compelling reason is that if you HAD an MLS OPE which happened to use a security kernel and a process per level, you'd need IT to be listening for incoming connection requests "from the net" rather than having the Host tell it to do so, for various esoteric reasons--but in order to cater to the possibility, we want the function in the protocol from the

beginning, on the grounds that we can envision SOME other uses for it even in non-MLS environments [unlike the security labeling trick discussed above, which only seems to make sense for MLS Hosts/OPEs--that is, it doesn't burden the Host to reject a Begin every once in a while but it would to go around labeling "chunks" unnecessarily all the time].)

Routing

Concern has been voiced over the issue of what provisions the protocol should make to deal with the situation where a Host, probably for traffic/load reasons, has multiple OPEs and the question arises of which OPE to use/route to. I claim this is a non-issue at the protocol level. If the Host-side H-FP PI gets a "No resources" response to a Begin, it can go off to another OPE if it wants to. "Not our department". The conceptual model is that of a Host and AN OPE--which "ought to" be expandable to carry more load at some level. If you want multiple links for some reason, the simplest solution would seem to be to have multiple Channel Layers as well, but the whole thing just gets too iffy to have anything sensible to prescribe in the protocol. In other words, extending the concept to deal with discrete multiple OPEs is either a Fabrication sort of thing, or a Notes to Host-side Implementors sort of thing on a per specific OPE basis.

Operator Interface

It's probably implicit in the foregoing, but it might be worth saying explicitly that the operator interface to a specific OPE is a non-issue in terms of the protocol, beyond the provision we're made for "Shutdown coming" responses as a reflection of a probable operator interface action we imagine most operator interfaces would provide. (It might also be worth noting that if your Host does "color changes", your OPE had better have a trustworthy way of being told to change the label it plops on all IP datagrams it emits, but that comes under the heading of an Aside to Specialized Implementors.)

Fine Points

There are a couple of known "loose ends" which are exceedingly fine points in some sense that do bear separate mention:

The Allocate Event

While mentally testing to see if the new H-FP would indeed off-load TCP, we came up against an interesting question: Viewing H-FP as "just an interface at a distance" to a TCP PI, what about the Allocate "Interface Event" in the TCP spec? As far as I'm concerned, this could be classed as a non-issue, because I submit that the spec is wrong in declaring that there is such a thing as a MANDATORY Interface Event whereby the user of a TCP PI lets the PI know how much data it can take. Granted, you might find such a thing in most implementations, but what if you were in a virtual memory environment with segment sharing (or a distributed supervisor) and you wanted to avoid copies, so all that passed at the interface to the PI (or even at the interface from the PI) was a pointer? That is, the "DOD version" of the TCP spec has fallen into the trap of assuming things about the execution environment that it shouldn't have.

One moral of this is that

> AN INTERFACE TO AN INTERPRETER OF A PROTOCOL IS N*O*T "THE PROTOCOL".

Another moral is that the interface to the Host-side H-FP PI is hard to say much about, but is where the equivalent functionality will be found if you've offloaded TCP. That is, it's reasonable to let the user "tell" the outboard PI at Begin time if big or small buffers are expected to be in play "net-ward" as part of the protocol, but the outboard PI is expected to deliver bits to the Host as they come unless throttled by the Channel Layer, or by some to-be-invented other discipline to force the OPE to buffer. (For present purposes, we envision letting the Channel Layer handle it, but nifty mechanizations of encouraging the OPE to "make like a buffer" would be at least looked at.) As a Fabrication issue, it is the case that "equity" has to be dealt with with regard to the use of the OPE's resources (especially buffers) across H-FP connections/channels, but that's a different issue anyway, touched upon in the final fine point.

Precedence

Clearly, the existence of a notion of Precedence in DOD protocols has to get reflected in the outboard PI's implementations. Just what, if any, role it has in the H-FP, per se, is, however, by no means clear. That is, if the Host doesn't take Begins from the OPE and is "full up" on the number of Server Telnet connections it's willing to handle, what should happen if a high precedence SYN comes in on the Telnet Well-Known Socket (in present day terms)? Probably the OPE should arbitrarily close a low precedence connection to make room for the new one, and signal the Host, but even that assumes the Host will always hurry to be prepared to do a new passive Begin. Perhaps we've stumbled across still another argument in favor of "Symmetric Begins".... At any rate, Precedence does need further study--although it shouldn't deter us from making "the rest" of the protocol work while we're waiting for inspiration on how to handle Precedence too.

A Note on Host Integration

The most important thing about Hosts in any intercomputer network is that they furnish the resources to be shared. The most significant obstacle to sharing those resources, however, is the fact that almost invariably they were designed under the assumption that the Host was a fully autonomous entity. That is, few operating systems currently deployed "expect" to be members of a heterogeneous community of operating systems. In many cases, this built-in insularity goes so far as to have applications programs cognizant of the particular type of terminal from which they will be invoked.

Intercomputer networking protocols attempt to resolve the problems of heterogeneity by virtue of presenting appropriate common intermediate representations (or "virtualizations") of the constructs and concepts necessary to do resource sharing. A Host-Host protocol such as TCP "is" a virtual interprocess communication mechanism; a virtual terminal protocol such as Telnet obviously is a mechanism for defining and dealing with virtual terminals; FTP offers common representations of files; and so on. It cannot be stressed strongly enough, though, that this entire approach to intercomputer networking is predicated on the assumption that the modules which interpret the protocols (PIs, as we'll refer to them often) will be PROPERLY integrated into the various participating operating systems. Even in the presence of powerful OPEs, wherein the bulk of the work of the various PIs is performed outboard of the Host, the inboard "hooks" which serve to interface the outboard PIs to the native system must not only be present, they must be "right". The argument parallels the analysis of the flexible vs. rigid front-ending attachment

strategy issue of [1]; to borrow an example, if you attempt to integrate FTP by "looking like" a native terminal user and the operator forces a message to all terminals, you've got an undetected pollution of your data stream. So the key issue in attaching Hosts to networks is not what sort of hardware is required or what sort of protocol is interpreted by the Host and the OPE (or comm subnet processor, for that matter), but how the PIs (full or partial) are made to interrelate with the pre-existing environment.

It would be well beyond the scope of this document to attempt even to sketch (much less specify) how to integrate H-FP PIs into each type of operating system which will be found in the DoD. An example, though, should be of use and interest. Therefore, because it is the implementation with which we are most intimately familiar, even though it's been several years, we propose to sketch the Multics operating system integration of the original ARPANET Network Control Program (NCP)--which is functionally equivalent to an H-FP PI for offloading ARM L II and L I--and Telnet. (A few comments will also be made about FTP.) Note, by the way, that the sketch is for a "full-blown" H-FP; that is, shortcuts along the lines of the scenario-driven approach mentioned above are not dealt with here.

One of the particularly interesting features of Multics is the fact that each process possesses an extremely large "segmented virtual memory". That is, memory references other than to the segment at hand (which can itself be up to 256K 36-bit words long) indirect through a descriptor segment, which is in principle "just another segment", by segment number and offset within the segment, so that a single process--or "scheduling and access control entity"--can contain rather impressive amounts of code and data. Given that the code is "pure procedure" (or "re-entrant"), a "distributed supervisor" approach is natural; each process, then, appears to have in its address space a copy of each procedure segment (with system-wide and process-specific data segments handled appropriately). Without going too far afield, the distributed supervisor approach allows interrupts to be processed by whichever process happens to be running at a given time, although, of course, interprocess communication may well be a consequence of processing a particular interrupt.

A few other necessary background points: A distinguished process, called the Answering Service, exists, originally to field interrupts from terminals and in general to create processes after authenticating them. Other shared resources such as line printers are also managed by distinguished processes, generically known as "Daemons". Device driver code, as is customary on many operating systems, resides at least in part in the supervisor (or hard core

operating system). Finally (for our purposes, at least), within a process all interfaces are by closed subroutine calls and all I/O is done by generic function calls on symbolically named streams; also, all system commands (and, of course, user written programs which need to) use the streams "user_input" and "user_output" for the obvious purposes. (At normal process creation time, both user I/O streams are "attached" to the user's terminal, but either or both can be attached to any other I/O system interface module instead--including to one which reads and writes files, which is handy for consoleless processes.)

All that almost assuredly doesn't do justice to Multics, but equally likely is more than most readers of this document want to know, so let's hope it's enough to make the following integration sketch comprehensible. (There will be some conscious omissions in the sketch, and doubtless some unconscious ones, but if memory serves, no known lies have been included.)

Recalling that NCP is functionally equivalent to H-FP, let's start with it. In the first place, the device driver for the 1822 spec hardware interface resides in the supervisor. (For most systems, the PI for H-FP's link protocol probably would too.) In Multics, interrupt time processing can only be performed by supervisor segments, so in the interests of efficiency, both the IMP-Host (1822 software) Protocol PI and the multiplexing/demultiplexing aspects of the Host-Host Protocol PI also reside in the supervisor. (An H-FP PI would probably also have its multiplexing/demultiplexing there; that is, that portion of the Channel Layer code which mediates access to the OPE and/or decides what process a given message is to be sent to might well be in the supervisor for efficiency reasons. It is not, however, a hard and fast rule that it would be so. The system's native interprocess communications mechanism's characteristics might allow all the Channel Layer to reside outside of the supervisor.)

Even with a very large virtual memory, though, there are administrative biases against putting too much in the supervisor, so "everything else" lives outside the supervisor. In fact, there are two places where the rest of the Host-Host Protocol is interpreted on Multics, although it is not necessarily the case that an H-FP PI would follow the same partitioning even on Multics, much less on some other operating system. However, with NCP, because there is a distinguished "control link" over which Host-Host commands are sent in the NCP's Host-Host protocol, the Multics IMP-Host Protocol PI relegates such traffic to a Network Daemon process, which naturally is a key element in the architecture. (Things would be more efficient, though, if there weren't a separate Daemon, because other processes then have to get involved with interprocess communication

to it; H-FP PI designers take note.) To avoid traversing the Daemon for all traffic, though, normal reads and writes (i.e., noncontrol link traffic) are done by the appropriate user process. By virtue of the distributed supervisor approach, then, there is a supervisor call interface to "the NCP" available to procedures (programs) within user processes. (The Daemon process uses the same interface, but by virtue of its ID has the ability to exercise certain privileged primitives as well.)

If a native process (perhaps one meaning to do "User Telnet", but not limited to that) wanted to use the network, it would call the open primitive of "the NCP", do reads and writes, and so on. An interesting point has to do with just how this interface works: The reads are inherently asynchronous; that is, you don't know just when the data from the net are going to be available. In Multics, there's an "event" mechanism that's used in the NCP interface that allows the calling process to decide whether or not it will go blocked waiting for input when it reads the net (it might want to stay active in order to keep outputting, but need to be prepared for input as well), so asynchrony can be dealt with. In the version of Unix (tm) on which an early NFE was based, however, native I/O was always synchronous; so in order to deal with both input from the terminal and input from the net, that system's User Telnet had to consist of two processes (which is not very efficient of system resources). Similar considerations might apply to other operating systems integrating H-FP; native I/O and interprocess communication disciplines have to be taken into account in designing. (Nor can one simply posit a brand new approach for "the network", because Telnet will prove to rely even more heavily on native mode assumptions.)

The other aspect of NCP integration which we should at least touch on--especially because process-level protocols make no sense without it--is how "Well-Known Sockets" (WKSs) work. In broad terms, on Multics the Network Daemon initially "owns" all sockets. For Well-Known Sockets, where a particular process-level protocol will be in effect after a successful connection to a given WKS, code is added to the Answering Service to call upon the NCP at system initialization time to be the process "listening" on the WKSs. (This is a consequence of the fact that the Answering Service is/was the only Multics process which can create processes; strategies on other systems would differ according to their native process creation disciplines.) How to get the "right kind of process" will be sketched in the discussions of the process level protocols, but the significant notion for now is that typically SOME sort of prior arrangement would be done by any networked Host to associate the right kind of process with a WKS.

Now, we don't expect that the foregoing will enable even the world's greatest system jock to go out and design the integration of an H-FP PI for a system that had never been networked (in the ARPANET style of networking) before. But we propose to stop there and turn to some comments on process level protocols, for two reasons: In the first place, it would take us much too far afield to go into significantly greater detail; and in the second place, because of the functional equivalence of H-FP and NCP combined with the number of operating systems which have integrated NCP and, for that matter, TCP/IP, which are also functionally equivalent to H-FP (used for offloading L II and L I), models are available in the ARPANET community and concerned H-FP PI implementors can follow them.

Turning to Telnet integration, and returning to Multics as an example, we note that "User Telnet" is straightforward. "All you need" (for small values of "all") from an INBOARD User Telnet is a command that gives the user some sort of interface, converts between the native Multics character set and terminal discipline and the Network Virtual Terminal equivalents (and as Multics is very generic when it comes to I/O, that's not hard), and writes and reads "the net" (more accurately, calls upon the Host-Host protocol PI--or upon the H-FP PI to get at the H-HP--appropriately). (One point that's not obvious: make the Well-Known Socket "on the other side" a parameter, defaulting to the Telnet WKS, because you'll want to use the same command to get at other process-level protocols.) If there's an OPE in play which offloads User Telnet, however, things can be even simpler: the inboard command just reads and writes the terminal and lets the OUTBOARD User Telnet PI handle the conversion to and from the Virtual Terminal form (presumably, from and to the desired local form).

When it comes to the incoming ("Server") aspects of Telnet, life can get complicated on some systems for an inboard implementation. However, fortunately for our purposes,

Multics' native mechanisms lend themselves readily to integration; an awareness of the inboard issues will be useful even if in response to a connection attempt on the Telnet WKS, the (Server) Host is obligated to associate the connection (the actual logic is somewhat more complex under the ARPANET Host-Host Protocol, which employs paired simplex connections) with a process that is prepared to translate between Telnet and native mode representations and otherwise "look like" a local user process--that is, in particular the connection becomes an I/O source/sink to the native command processor on time-sharing systems. As indicated, process creation is taken care of in Multics by having the Answering Service process listen on the WKS. Because the Answering Service is in some sense

just another Multics process, it too does user I/O through the normal system mechanisms. So while for local terminals the user I/O streams are attached through a module called "ttydim" (where "dim" stands for "device interface module"), NVTs are attached through a functionally equivalent and identically invoked module called "nttydim" (the Answering Service knows which DIM to use based on the symbolic designator of the "line" on which it received the interrupt, as it happens).

[The notion of "attaching" the streams bears a bit more explanation: Attach is a primitive of the Multics generic I/O mechanism which associates a stream name and a particular DIM (or I/O system interface module in later terminology); the other I/O primitives (read, write, etc.) are invoked with the stream name as a parameter and an I/O "switch" causes the entry point corresponding to the primitive to be invoked in whichever DIM the stream is currently attached to. So a Server Telnet process starts life attached through nttydim to a particular network connection, while a local process starts life attached through ttydim to a particular physical line, and both processes proceed indistinguishably (viewed from outside the I/O switch, anyway).]

The pre-existing orderliness that makes things easy on Multics does not, unfortunately, appear in all operating systems. Indeed, delicate choices occasionally have to be made as to WHICH native terminal to map to on systems that don't do generic I/O in native mode, and it is likely that for some systems the particular mapping to bring into play in Server Telnet might be determined by the particular application program invoked. This issue can become very touchy when the application "expects" a "data entry terminal", say. The Server Telnet for such a system would naturally attempt to negotiate the "DET" option with the corresponding User Telnet. But the user might be at a physical terminal that isn't a member of the DET class, so that User Telnet must either refuse to negotiate the option or--and we would recommend this alternative strongly, as it seems to be within the "spirit" of the protocol--offer some sort of simulation, however crude, of the behavior of a DET. Also, something sensible has to be done on systems where there is no clear analog of the command processor expected to be managing the Server process. (Say, when a "menu" of applications is always displayed on an available terminal in native mode.)

A final Telnet integration issue (although other points could be noted, we're not pretending to be exhaustive and this should be enough to "give the flavor"): The Telnet Interrupt Process generic function calls for particularly careful integration. Here, the intent of the function is to virtualize what is called the "quit

button" on some systems. That is, the user wants the system to interrupt his process (which may, for example, be in a loop) and get back to the command processor (or "the system" itself). On native character-at-a-time systems, the native mechanism is usually the entering of a particular "control character"; on native line-at-a-time systems, the native mechanism is usually the striking of the "ATTN" or Interrupt button or the "Break" key (sometimes more than once, to distinguish it from a communication to the executing program). But the native mechanisms typically involve interrupt time code, and Server Telnet typically wouldn't be executing at that level, so the solution (omitting the intricacies of the interaction with the NCP or the H-FP PI, which also get into the act) would be to make use of--in the Multics case--a pre-existing INTRAprocess signal, or to add such a mechanism (unless the architecture chosen has a Server Telnet Daemon of some sort, in which case an INTERprocess signal would be needed).

The extension of the foregoing to an outboard Server Telnet may not be obvious, but we won't expend a great deal of time on it here. Even if "the protocol" is being handled in an OPE, the Host-side software must be able to associate an H-FP connection with the command language interpreter of a user process and to respond appropriately to an H-FP Signal command if it arrives, and the OPE must know not only the desired character set but also the local equivalents of Erase and Kill, at the minimum.

We'll skip FTP integration, on the grounds that this note is already too lengthy, except to mention that in the OUTBOARD case it's still going to be necessary to convey the name of the appropriate file and directory to/from some appropriate Host-side code. (Similar problems must be dealt with for outboard handling of "mail" if it's not part of FTP.)

One other "integration" issue, which has been hinted at earlier and about which not much can be said beyond some general guidelines: The "top edge" of a Host-side H-FP protocol interpreter (i.e., the Host user program interface, for

Hosts that are "doing real networking" rather than just using the OPE to get at User Telnet and/or FTP and to offer Server Telnet and/or FTP [and maybe "mail"], presumably in the "scenario-driven" fashion sketched earlier) MUST BE APPROPRIATE TO THE HOST. In other words, on Multics, where "everything" is closed subroutines, there would presumably be a closed subroutine interface with event channels for reads, pointers to buffers, and all that sort of thing, but on some other style of operating system, the interface to the H-FP PI might turn out to be "all" interprocess communication, or to "look like" a

device of some special class, or "all" system calls/JSYSs/EOTs/Whatevers. We can't be much more specific, but we'd be remiss to convey any impression that H-FP is a "free lunch". As noted, an H-FP PI requires the same kind of integration as a generic NCP--it's just smaller, and serves as insulation against changes (in the offloaded protocols in general, or in the proximate comm subnet in particular).

References

(References [1]-[3] will be available in M. A. Padlipsky's "The Elements of Networking Style", Prentice Hall, 1985.)

[1] Padlipsky, M. A., "The Host-Front End Protocol Approach", MTR 3996, Vol. III, MITRE Corp., 1980.

[2] Padlipsky, M. A., "The Elements of Networking Style", M81-41, MITRE Corp., 1981.

[3] Padlipsky, M. A., "A Perspective on the ARPANET Reference Model", M82-47, MITRE Corp., 1982.

[4] Bailey, G., "Network Access Protocol", S-216,718, National Security Agency Central Security Service, 1982.

[5] Day, J. D., G. R. Grossman, and R. H. Howe, "WWMCCS Host to Front End Protocol", 78012.C-INFE.14, Digital Technology Incorporated, 1979.