Danny

IEN-73

N. Abramovitz
M. Padlipsky
K. Biba
23 January 1979

A TCP Implementation Issue

INTRODUCTION

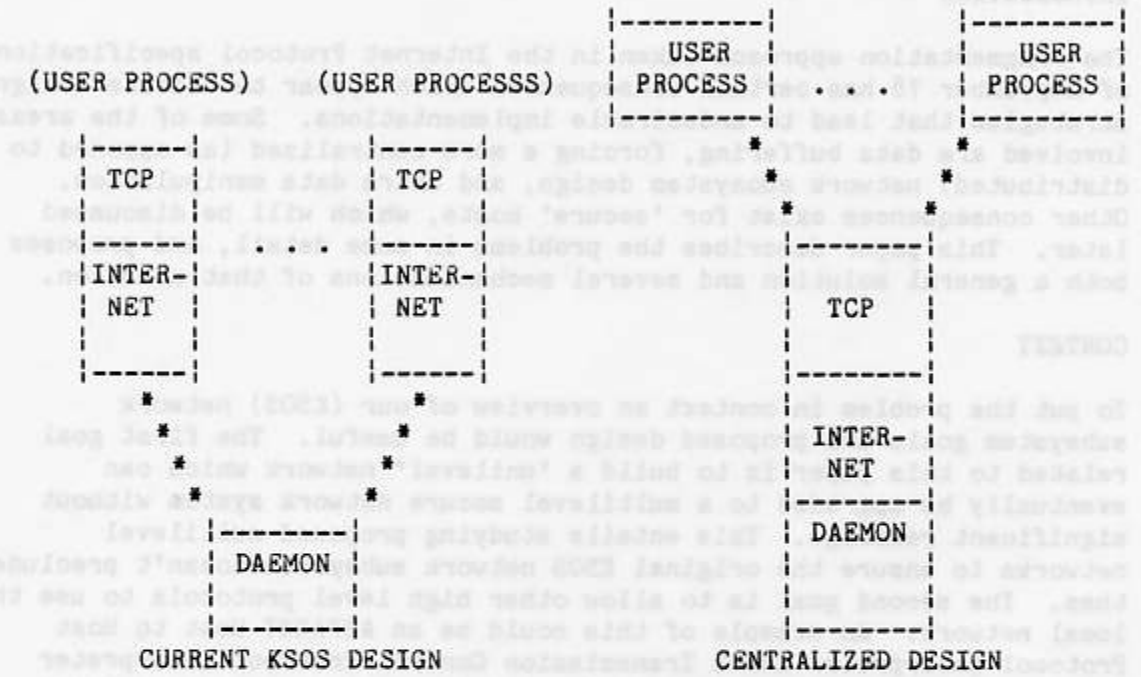
The fragmentation approach taken in the Internet Protocol specification of September 78 has certain consequences which appear to dictate design strategies that lead to undesirable implementations. Some of the areas involved are data buffering, forcing a more centralized (as opposed to distributed) network subsystem design, and extra data manipulation. Other consequences exist for 'secure' hosts, which will be discussed later. This paper describes the problems in some detail, and proposes both a general solution and several mechanizations of that solution.

CONTEXT

To put the problem in context an overview of our (KSOS) network subsystem goals and proposed design would be useful. The first goal related to this paper is to build a 'unilevel' network which can eventually be upgraded to a multilevel secure network system without significant redesign. This entails studying proposed multilevel networks to ensure the original KSOS network subsystem doesn't preclude them. The second goal is to allow other high level protocols to use the local network. An example of this could be an ARPANET Host to Host Protocol interpreter and a Transmission Control Protocol interpreter communicating through a single IMP. A third goal is to minimize the number of data copy operations. For machines (like the PDP-11/70) with limited virtual addressing, this forces certain design constraints. The last important goal is to minimize the 'denial of service' threats such as a malicious user monopolizing all the network buffer space.

From the design goals above, and others not germane here, we decided to choose a basically distributed design over a more centralized one. In our view the key difference between distributed and centralized is that there are many logical copies of the protocol handlers instead of just one. Therefore, a distributed implementation would prefer to demultiplex data sooner than a centralized one. The KSOS network subsystem has the Transmission Control Protocol and Internet Protocol handlers on a per user process basis. The distributed protocol handlers communicate with a single process controlling access to the network hardware device. (This single process, the Daemon, could also be

distributed, but synchronization on the network device would be necessary and we would rather avoid that.) Buffer space is allocated in each distributed handler to eliminate denial of service to all network users on a host by common buffer space exhaustion by a single user. Also i/o can now be done directly into user buffers thereby decreasing data copy operations. The following diagrams show the KSOS distributed design and also a centralized design for comparison.



DIGRESSION ON SECURITY

Note that it is an important pragmatic goal of our design to minimize the amount of code in the Daemon. This is so because the Daemon is currently the only network subsystem component responsible for security and thus the only network subsystem component we intend to subject to verification. The secure systems theoretic point here is that if the multiplexing/demultiplexing of data to/from a network is performed securely, we can be spared the agony and expense of verifying all the network code and still be in a position where security compromises will not be a problem - even if the unverified protocol interpreters 'in' the participating process can foul up the individual process, that is, they can't 1) bring down the whole network attachment or 2) cause data to be compromised. Admittedly it would be special pleading to ask for a protocol change just to make our Daemon smaller, but as we will show

there are practical problems for non-secure systems as well when it comes to the sort of demultiplexing implied by the current Internet Protocol fragmentation approach.

ANALYSIS

This whole discussion is based upon our current understanding of fragments. A fragment consists of an Internet header and some amount of data. This data must be in multiples of 8 octets for the first n-1 fragments. TCP headers then may span fragments and many fragments may not have a TCP header at all. Since a connection is uniquely defined by a tuple of sending/receiving network number, host number, and port number and port numbers are contained in the TCP header, most fragments will arrive without an adequate identifying characteristic for the Daemon (or whatever the demultiplexer is on some other system) to decide which user buffer area should receive the fragment. The current Internet Identification field is assigned by the sending side, so it is priori inappropriate for demultiplexing at the receiving side. Even if it were gotten from the receiver side via some sort of initial connection protocol, the Daemon could not demultiplex on it anyway, because it doesn't know when this field will stop being used without going through Internet Protocol handling; that is, the Identification field only helps in fragment reassembly, by its very nature. So, if matters remain as they are today, the Internet Protocol forces itself into the Daemon.

What are the implications of placing the Internet Protocol inside the Daemon? From a practical point of view, we can think of two problem areas: code space and data buffering. (We will also address some security problems.)

On many systems the functions of the Daemon occur down in the guts of the operating system. On systems without virtual memory or systems tight on 'monitor' space, it may be difficult to add more code to the system. With the distinct possibility of greater functionality being given to the Internet Protocol via various options, this code space limitation may become a real problem. As a sidelight, testing new versions of the Internet Protocol interpreter could be quite messy.

The major practical consideration is the buffering strategy to be used for the Internet Protocol handler. As stated previously, fragments can not be given to the distributed handlers until all the fragments for a segment arrive. This fragmentation approach mandates the use of a centralized buffering strategy for collecting fragments for all network users. For machines with limited addressing range (the PDP-11/70 and its contemporaries), the amount of centralized buffer space dictates the number of network users able to use the network.

Limitations with the number of users is not the only problem, however: let's examine the buffer space partitioning implications of the implicitly dictated centralized strategy: the buffer space could be addressable by all network users and the Daemon by using shared memory, or the buffer space could be located internal to the Daemon. For the case that the buffer space is shared between all users, the Daemon must be informed by the user that a buffer is ready to be reused. An obvious consequence could occur: a malicious user could exhaust the buffer resource. This problem could be combated by limiting the number of buffers a network user may use, but this approach has the disadvantage of decreasing effective throughput. The other case of only letting the Daemon have access to the buffer area has all the problems of sharing the area, plus the additional one of data must be copied to the network user space. Another factor to slow down effective throughput.

From a security view point we make these claims: First, as stated previously, the more code we need to include inside the security perimeter the harder it will be to verify the correctness of the Daemon. Second, we suspect the Internet Protocol specification should also be verified before secure systems could use it. Third, we feel we can not ignore the security problems caused by centralized buffering: we can not have global shared memory between the Daemon and all network users; this would be a violation of the security model because users can be at different levels. Further, denial of service by buffer exhaustion is unacceptable in our view of a secure system.

GENERAL SOLUTION

In light of the above observations, we feel that the current fragmentation approach is impractical for more operating system and machine architectures than just our own. It is of course, particularly distasteful to us. We feel the general solution is to have the Internet header contain a field which allows demultiplexing to take place on each fragment. Exactly what the field should contain involves both matters of taste and sundry more or less subtle tradeoffs, which we will address next. The first major point we wish to make is that the ability to demultiplex fragments directly in order not to preclude a distributed implementation (or implicitly legislate a centralized implementation) is necessary in principle.

MECHANIZATIONS

One way to mechanize the general solution is by MOVING the port number fields in the TCP header into the Internet header. Because port numbers are uniquely associated with processes, this would allow for the desired demultiplexing. Unfortunately this approach not only increases the size of the Internet header, but it also implicitly changes the scope of the

Internet Protocol from an Internetworking-oriented datagram service to having at least some aspect of a process to process protocol. This approach may well also violate the principle of protocol layering, which we are reluctant to do. That is, TCP really ought to be usable without Internet for strictly local nets and, for that matter, for nets which opt for some reason to do internetting via 'smart' gateways. Further it would require modifications of both Internet and TCP and changes to current software as to where connection validation would be performed as well as minor data structure modifications.

A more desirable mechanization is to REPLICATE just the destination port in the Internet header. This again increases the size of the Internet fragment, but at least USE of this field would be at the implementer's discretion, and of course, the increase is smaller. The impact on existing software should be less than the previous method. Indeed, existing Internet Protocol handlers wouldn't need to change their logic flow. Also it appears that layering would be preserved, because it's not really 'the' port number in the field, it's just a convenient connection unique number on which to demultiplex.

Two other mechanizations occur to us, both of which seem to be sufficiently undesirable in principle that we note them only to prove that we have not overlooked them: In the first place, we could establish some sort of convention which uses the 1822 Specification link/message-id to demultiplex on. In the second place, we could invent some entirely new number which would be assigned by the receiving host as it likes, to live in the Internet header. The trouble with both of these schemes is that they would entail noticeable expansion to existing protocols, and for this reason we do not favor either. (If forced to choose, the 'new' number would clearly be preferable because it is not communication subnet specific, but we do not really feel we need to do what amounts to the invention of an initial connection protocol on the fly just to solve the demultiplexing problem.)

RECOMMENDATION

We feel there is a need to be able to demultiplex on fragments, for pragmatic and for security related reasons. We do not contend that we have covered all possible solutions to this problem, nor all possible mechanizations of our own solution. We lean towards the second mechanization on the principle that it leaves the multiplexing/demultiplexing issue up to the TCP implementers. We request comments concerning to the points raised in this paper. Assuming the comments don't point out something fundamental we have inadvertently overlooked, we recommend that the Internet Protocol header for each fragment be expanded to include a field which contains a

replication of the TCP destination port field of the TCP header for the associated segment.