IDN-1

Internet Access Protocol

Part 1

Eric Rosen

July 1981

I've been thinking about the internet access protocol and the stuff we need in the gateways to handle it. I'm going to free associate a bit, in the hope of getting some feedback.

In AUTODIN II, the source nodes maintain structures called "Bookkeeping Blocks (BKBs)." Each source node has one bookkeeping block for each host-host "connection". The main purpose of these is to maintain the information needed to enforce flow control on a host-host basis. These structures are also useful for accounting and instrumentation purposes. I think we will need to have similar structures in the source gateways. Perhaps we can call them "connection blocks", though this particular term may be politically unsuitable. These blocks would define the granularity of flow on which we do flow control. If there is one block per source/destination host pair, then we

will be able to control the flow between each host pair, without any interaction with flows from the same source host to other destination hosts. We could instead keep one block per source host, but then we would be unable to throttle any one host-host flow without also throttling all other flows from the same source host.

The trade-off here is in the amount of fine grain we have in throttling flows vs. the amount of overhead (memory, and to some degree, cycles) we need to maintain the additional information needed to make the grain finer. The flow control with the finest grain might be based on the notion of a flow between a source/destination pair of PROCESSES; but it's not clear that we will be able to make such fine distinctions in our flow control algorithms, and the overhead might be very large. The coarsest grain might come from using the notion of a source/destination pair of gateways. That results in much less overhead, but provides no fairness among different flows between the same pair of gateways. Applying flow controls on a host-host basis is probably the best compromise.

Actually, I should be more precise about what I mean by "host-host." Host-to-host in this context can mean logical-

address-to- logical-address, physical-address-to-physical-address, or some combination (such as defining a "host" to be a particular logical/physical address pair.) I think logical-address-to-logical-address might be the best way of identifying flows for the purposes of flow control, but that might give an advantage to hosts which happen to have a lot of logical addresses. Rather than setting this in concrete based on purely a priori considerations, perhaps we should make our system flexible enough so that we could at any time (by altering the setting of a parameter and then restarting) change the way we set up the connection blocks (and hence the way we individuate flows for flow control purposes.)

You are probably most familiar with the sort of flow control based on a windowing scheme, as in TCP. Besides the fact that windowing is very difficult to apply to datagrams, I don't like it anyway, because it is too prone to interactions. Windowing schemes attempt to use a single mechanism, that of sequence numbers, for three different purposes: flow control, error control, and sequentiality. The use of a single mechanism for three disparate purposes is bound to introduce strange interactions in the inevitable cases in which no single action

can serve each of the three separate purposes. In the internet, we need to have flow control, we would like to have error control, and we don't want sequentiality (at least, not always), so windowing based on sequence numbers would be particularly inappropriate. I would like to suggest the following scheme for enforcing flow control. Each connection block should contain a number of packets P and a number of bits B such that no more than P packets per unit time or B bits per unit time be accepted by the gateway on that particular host-host flow. If a source host tries to exceed this rate, any packets it sends in excess if the rate will be explicitly NAKed. The NAK will contain enough information to uniquely identify the NAKed packet and will also inform the host as to the maximum rates at which it can send. NAKed packets will be discarded by the gateway. (Of course, we retain the option of not sending NAKs if we are out of resources; that is, if necessary, we can just drop packets with no notice, but NAKing when possible is better.)

Hosts should always be able to find out what flow control limits are being imposed. So we need to have a control message which a host sends to a gateway to find this out; the gateway will reply if possible. An option: let the host set a bit in the

internet header of a data packet to ask the source gateway to explicitly acknowledge its receipt of the packet. The ACK can contain any flow control restrictions, and hosts can set this bit every so often, just to see whether the source gateway is really taking their packets. (The source gateway might want to refrain from sending these ACKs until it forwards this packet on its next hop. This delays the sending of the ack, but gives it somewhat more meaning.)

This flow control mechanism gives us a means for enforcing flow control which is independent of any particular flow control algorithm we happen to use within the internet. Information from the routing algorithm, from specific congestion control techniques, and from our monitoring of local resource utilization (e.g., buffer space in this gateway itself) can all be mapped into the rate limitations that we code within the connection blocks, and we can experiment with the mapping functions to our heart's content, without any need to modify host software.

Since we will have only a finite number of connection blocks, we will have to recycle them. We can recycle a block if its connection has been idle for a certain amount of time. If a

packet comes in from source logical address S for destination logical address D, and there is no S-D block currently in use, one must be allocated. If none are free and available for allocation, we will have to NAK the packet. This might aggravate the user, but is just a consequence of the need to have flow control, and hence to maintain information about flows. Note that since we are keeping flow control information in these blocks, we don't want to be too free about recycling them, or we lose important information.

One problem that we often seem to run into with such data structures is the inability to efficiently index them in any of several different ways. That is, we might want to be able to efficiently access all blocks for a particular destination host, or all blocks for flows going to a particular destination gateway, or something else. We want to be able to do this without having to do a linear search of all blocks whatsoever (which would probably include even unused blocks.) One possible way to handle stuff like this is to make the blocks be queue items on any of several different queues (i.e., on several simultaneously). Santos' NMFS handles this rather naturally, but I don't know what CMOS would think of it.

We might consider having some error control on the host access protocol. It would be nice if we could require the host to put in a software checksum; we could discard the packet if the checksum is incorrect. I think an access Pathway checksum of this sort is really needed, and should probably be more powerful than the silly additive checksum usually used, about whose power I have seen no mathematical results. (Santos once tried to convince me (jokingly?) that the additive checksum used in the ARPANET is sufficiently powerful, on the grounds that in all the years of the ARPANET's existence, not one undetected error has ever been detected. I wasn't convinced. However, probably any checksum which hosts are required to do will have to be trivial.) It might be nice to give the host the option of putting on a checksum or not, except that I can't think of any way of implementing that. (Suppose we assign a bit in the internet protocol header to indicate that the host did not put in a checksum. Then if this bit is on, does it mean that the host did not put in the checksum, or that the bit was turned on spuriously?)

At any rate, we will need an end-end checksum (purely within the internet system, i.e., between the source and

destination gateways), which we should probably make more powerful.

We will have to keep measurements of the utilization of the connection block resource. I like to do the following measurement: when a request for a new connection block is made, increment by one a counter of requests, and then find out how many blocks are free and available, and add that number to another counter. Then dividing the second counter by the first yields the average number of blocks available per request, which is a good measure of whether the utilization is high, and of whether we are growing short. We should also keep track of the rate at which blocks get recycled. There are some measurements which we could keep in the blocks themselves: packet and bit rates on the "connection", packet size histograms, number of packets for each protocol number, number of packets in each service class, etc. It might be convenient to keep such counters right in the blocks. Or we can keep pointers to the counters in the blocks, in which case several blocks can share counters, for a somewhat more coarse measurement. (This might be necessary to save space, if, for example, pointers are single words but counters are double precision.) Or we could have each block

contain a pointer into a fixed length measurement area, with the different counters being identified by their offsets into the measurement area. It is probably a good idea to work out some such general and flexible measurement scheme in advance, so that we don't need new ad hoc designs every time we think of an additional measurement.

In the current internet, the same protocol is used among gateways as is used among hosts and at the gateway-host interface. In the ARPANET, the protocol used at the host-IMP interface is all contained in the 1822 leader, and the protocol used between IMPs is contained in the packet headers. The actual leader is never carried across the network, but the header (which does not resemble the leader in format) contains enough information so that the leader can be reconstructed before the packet is passed to the destination host. Packet headers contain somewhat more information than the leaders do, but hosts never see the headers. I think we will need this distinction in the internet (which is one reason why we need our own end-end checksum). Here are some information fields which will have to be in the header, but not the leader:

-Physical address of the source gateway. This will be the node number of the source gateway within the numbering scheme of the internet. We need this for returning end-end control messages, such as DNA messages. It is also good to have such a field for debugging purposes. Yet this number may not even be known at the host level.

-Physical address of the destination gateway (needed for routing).

-Packet type. Is this a control message (routing, DNA, etc.) or a user-supplied data message. By "control message," I refer only to messages which are used for internal internet control purposes.

-The "ACK this packet, neighbor" flag. Remember that the delay measurement procedure I proposed for the routing algorithm requires that we be able to select certain packets for hop-hop acknowledgments.

-Compatibility/version flags, so that we can distinguish among different versions of our stuff.

-A whole bunch of spare bits.

-If we have WWVB radio clocks available we might want to keep a couple of (optional?) words for timing purposes.

-End-end checksum

The following is the information that we need in the user-specified leader (all of which must also be in the header):

-Source host logical address. This should probably be verified by the gateway; if a certain logical address is not recognized by a gateway as that of a local host (i.e., a host that it knows how to reach "directly," without using the internet), the the packet should be discarded.

-Destination host logical address.

-Checksum for access Pathway only

-Service class (including priority, precedence, etc.) We'd better allocate a bunch of spare bits here.

-Packet identifier

-Protocol number

-Any garbage needed to enable fragmentation/reassembly

-Instrumentation flags. I am thinking of things like the trace bit in the 1822 leader, which when set causes a packet to be traced on its path through the network (if the trace package is loaded). I anticipate designing a trace package for the internet similar to the one we have in the ARPANET, and we need a bit that can be set at the host level. (We probably don't want users to be able to set this bit without explicit authorization, but we need to allow our fake hosts, such as message generators, to produce messages with this bit set.) Another possible instrumentation bit might correspond to the "tagged packet" bit of the ARPANET, which causes a packet (with no data) to have its data part filled with the identifier of each Switch it passes through, as well as the delay from that Switch to the next. Another possibly useful sort of instrumentation might be a selective acknowledgment bit, which would cause an ack to be sent to the source

gateway from either the destination gateway or from every intermediate gateway also. This might be useful for timing.

-A flag which asks the source gateway to ack this packet and furnish flow control information also.

We also need a special host-gateway control message by which a host can ask the source gateway for its delay (in ms.) to the destination gateway which it will use for a particular specified destination host, as well as a reply for the gateway to send back to the host.

I'll continue these thoughts in a subsequent message.