Link Level Protocols Revisited

Phil Kam, KA9Q

Brian Lloyd, WB6R QN

ABSTRACT

The LAPB protocol on which the connected mode of AX.25 is based was originally designed for point-to-point wire links, not shared multiple-access radio channels. This paper discusses the deficiencies of LAPB in the radio environment and suggests several improvements. These) include the simple (adjusting existing TNC parameters), the moderate (upward compatible implementation changes) and the radical (replacing LAPB altogether with a simpler and inherently much more efficient protocol).

We believe that these approaches deserve serious attention by the amateur packet community. The suggested AX.25 congestion control techniques should be used as soon as possible in existing networks, while the new "ACK-ACK" protocol should be considered in the design of backbone networks and eventually user-network links.

1. AX.25 in Review

The Amateur Radio Link Layer Protocol AX.25 [1] actually consists of two distinct sublayers. The upper sublayer is a connection-oriented protocol almost identical to the Link Access Procedures Balanced (LAPB) from CCITT X.25 [2]. Prepended to the LAPB control field in AX.25, however, is a special address field containing ASCII-encoded amateur radio callsigns and substation IDs (SSIDs). Each AX.25 frame contains, at a minimum, the callsign/SSID of the sender and intended receiver of the frame. Optionally, it may also contain a sender-specified list of digital repeaters, or digipeaters, through which the frame should be routed to its destination.

This additional lower sublayer contains most of the changes made to enable it to function in an amateur radio environment. An AX.25 address header always contains full source and destination addresses. Therefore it meets the definition of a "datagram" protocol, and we will call this field the "datagram sublayer" of AX.25.

The LAPB (upper) sub-layer of AX.25 belongs to the general class of ''ARQ'' (automatic request-repeat) protocols. It uses receiver acknowledgements (''ACKs'') along with sender timeouts and retransmission to increase the reliability of the service provided to higher layer protocols above that of the raw datagram sublayer.

The performance of an ARQ protocol depends heavily on how closely its design assumptions are met in practice. LAPB was designed for a full-duplex, point-to-point channel with low bit error rates; thus it performs poorly when used on a half-duplex, multiple-access radio channel with a high error rate. The remainder of this paper will discuss the special requirements of the amateur environment, analyze LAPB with respect to these requirements, and propose several alternatives. These range from a set of backward-compatible provisions to improve the LAPB retransmission algorithms in the presence of channel congestion, to the specification of an entirely new, connectionless link level control protocol that is both easier to implement (especially in the "multiple connect" case) and inherently much more efficient than LAPB.

2. Packet Radio Protocol Requirements

A good packet radio protocol must do four things:

1. Deliver the user's data as reliably as possible.
2. Deliver the user's data as quickly as possible.
3. Use the shared channel capacity as efficiently as possible.
4. Be reasonably easy to implement.

Naturally, these goals conflict. For example, on a full-duplex, point-to-point channel, the third consideration does not apply. The protocol is free to use as much transmission bandwidth as it chooses to maximize the other criteria, since it would otherwise go to waste. On a shared packet radio channel, however, a socially well-adjusted protocol should attempt to use as little channel bandwidth as possible to move a given amount of data, even at the expense of increased user delays. As we will see, this makes several features of LAPB undesirable.

3. Sliding Windows vs Stop-and-Wait

LAPB is a *sliding window* protocol. This means that it is possible to send more than one frame before an ACK is received for the first outstanding packet. The maximum number of frames that may be transmitted before the sender must stop and wait for an ACK is known as the window size. For the standard form of LAPB used in AX.25, the window size cannot exceed 7. Most implementations allow the user to decrease this limit further (e.g., the TAPR MAXFRAME option).

3.1. Pipelining

Sliding window protocols are popular because they allow the full utilization of high speed, *full duplex* circuits with long round trip delay. For example, a satellite path has a round trip delay of .5 seconds. If only one frame can be sent at a time, the transmitter must remain idle at least .5 seconds before an ACK could return, enabling it to transmit another frame. While the effect of this delay can be reduced by making each frame very long it cannot be eliminated unless more than one frame can be "in the pipe" at a time.

Pipelining causes major problems, however when the channel is half duplex. Since the receiver usually attempts to acknowledge incoming data as soon as the channel becomes clear, it is likely to collide with the sender if the sender attempts to send additional data without first waiting for the ACK for the data already sent.[1] Pipelining on a half duplex radio channel is therefore a. good example of a socially undesirable technique that attempts to reduce user delay at the expense of overall channel efficiency.

3.2. Transmission Efficiency

Given that a user wants to send a certain amount of data, a packet radio controller must control several parameters to packetize that data for transmission most efficiently.

Since it operates on a half-duplex channel, the controller must first decide how much to send on each transmission before switching to receive and waiting for an ACK. Sending more on each transmission reduces channel overhead by allowing each returning ACK to "cover" more data. It also increases the maximum throughput attainable on a channel with a given propagation dellay, since this rate can never exceed one window full per round trip interval. On the other 'hand, smaller transmissions are smaller targets for noise and interference.

Second, once it has decided to send a certain amount of data in a transmission, the data must be divided up into one or more consecutive frames.[2] The fewer frames used, the smaller the effects of frame overhead. On the other hand, a single large frame is unusable even if a single bit error occurs near the end, so sending the data as a burst of smaller frames might allow the receiver to "salvage" at least the beginning of the transmission. (Note that the "go-back-N" recovery strategy of LAPB requires that all frames received after an error be discarded and retransmitted, even if the later frames are received correctly).

The sender therefore controls three parameters: transmission size, frame window

[1] Note that TNCs capable of "multi-connect" operation (i.e., able to manage simultaneous connections with several different stations on the same channel) should allow unacknowledged data on only one connection at a time.

[2] Although pipelining is to be avoided, more than one frame may be sent in a single transmission (i.e., without interrupting carrier between frames).

size and maximum frame size, and setting any two determines the third. Unfortunately, values that improve performance in the presence of noise conflict with those that minimize overhead. Therefore, it follows that there should be an optimal set of parameters for a given channel when operating at a given error rate and with a protocol that has a given header and ACK overhead.

This is indeed the case. A program[3] was written to compute analytically the overall expected efficiency of an AX.25 transmission, given the following parameters:

1. A Gaussian bit error rate (each bit error being independent of all others, as would occur with "white" thermal noise).
2. A transmission size in bits.
3. The number of frames into which to divide the transmission.
4. The overhead in bits of a frame header and an ACK.

For all bit error rates studied (10^{-2} to 10^{-7}, ranging from a link that is almost completely unusable to one so good that almost anything works), maximum overall efficiency was always obtained when each transmission was limited to one frame. As expected, the optimum transmission (and frame) size increases with decreasing bit error rate. The efficiency of a large (and suboptimal) transmission can be improved by dividing it up into several consecutive frames. However, this is always less than the efficiency attained when the message is divided up into smaller single-frame transmissions, even when the extra ACK overhead is taken into account. In other words, a "stop-and-wait" (i.e., window size one) protocol with an appropriate frame size always uses the channel more efficiently than a sliding window protocol with a window size greater than one.

The "real world," though, is a bit harder to characterize because many (if not most) errors are caused by collisions rather than insufficient S/N ratios. Errors are therefore much more likely to occur in bursts, with entire transmissions often rendered useless. Collisions are harder to model than thermal noise, so their effect on the efficiency of a sliding window protocol is harder to evaluate. However, it seems reasonable to argue that the "salvage value" of a multi-frame transmission that has encountered a collision is likely to be even less than one corrupted by thermal noise. This is because in CSMA, collisions tend to occur near the beginning of transmissions, during the "collision window" represented by the time needed for one round trip propagation delay. Since a hit in the first frame of a transmission renders all later frames unusable, dividing up the transmission into multiple frames simply increases header overhead without improving net performance.

Since we have shown that sliding windows are suboptimal for our application, and because they account for much of LAPB's complexity, the first of our motivations for scrapping LAPB altogether and designing a new protocol better tailored to the radio environment becomes apparent. However, we recommend that current AX.25 TNCs be operated in a stop-and-wait mode. Simply set MAXFRAME to 1 and take steps to adjust the packet size to a value appropriate for the channel. This ensures that new data can never collide with a returning ACK, regardless if the implementation already enforces a "single outstanding transmission" rule. One implementation strategy that might be useful here is to hold additional outgoing data in a buffer until any previously sent data has been acknowledged, and then send as much as possible in the next data packet subject to the maximum packet size limit. This is much more efficient than "pre-packetizing" outgoing data according to special characters in the data (e.g., carriage return), since short lines would otherwise result in small transmissions and poor throughput. A similar strategy, the Nagle Rule [10] has become widely accepted by implementors of the ARPA Transmission Control Protocol (TCP), the most common transport (level 4) protocol used in the ARPA Internet [6,7].

One final issue, that of "Selective Reject," becomes moot when LAPB is operated in a stop-and-wait mode. If only a single frame can be outstanding at any one time, there is no need for this special measures. In any event, simulations have: shown that Selective Reject actually *degrades* performance relative to "standard':' LAPB, especially on links with large window sizes, when errors occur in bursts (as they often do in the real

[3] See Appendix 1 for the derivation of the formulas used in this program.

world). [9] The same paper proposed an alternative known as "multi reject" but this is again unnecessary if the window size is one frame.

4. Congestion Collapse

Because LAPB made no provisions for channel sharing, as now implemented on amateur packet radio TNCs it is extremely prone to channel "congestion collapse," a stable condition where virtually every transmission results in a collision. Congestion collapse can be avoided only when TNCs become able to adapt themselves dynamically to the load offered by other TNCs. This section proposes congestion control algorithms that should be made part of the formal AX.25 protocol specification. They will be of little help unless widely implemented, because there is little incentive (other than altruism) for one to use them unless everyone does.

Several factors combine to cause our currently severe congestion problems. The first is *jump-on,* where several stations with traffic to send collide with each other the instant the channel becomes free. The second and more serious problem is caused by *hidden terminals.* These are stations unable to hear (and defer to) transmissions from certain other stations. When this happens, the CSMA (carrier sense multiple access) algorithm breaks down and collisions become very frequent. Kleinrock [3] has shown that just one or two hidden terminals in a packet radio network are often enough to degrade performance below that of slotted Aloha, and it doesn't take many additional hidden stations for performance to approach that of pure Aloha. Gower and Jubin [4] observe that carrier sensing actually makes things *worse* in the presence of hidden stations because it aggravates jump-on.

A more fundamental problem with packet radio is that all dynamic multiple-access schemes (regardless of carrier sensing, hidden terminals, delay and so forth) exhibit a "negative resistance" characteristic at high load levels. As the offered load increases, the collision rate increases and usable throughput decreases. While the exact point at which optimum throughput is obtained on a given channel varies widely depending on the algorithm and the station characteristics, there is always the need to control the offered load dynamically if it is to be operated efficiently. Unfortunately, this is not true for current AX.25 implementations.

4.1. Retransmission Backoff

The most important change that must be made to stabilize the AX.25 protocol is for successive retransmissions caused by the nonreceipt of an ACK to be spaced out further and further over time. T'his technique is called *backoff,* and many variations exist.

The most well-known example is *binary exponential backoff,* used in the Ethernet local area network. [5] In Ethernet it is possible to detect a collision during transmission. When this occurs, the transmission is aborted and a backoff timer is set to a random value distributed evenly between zero and a number that doubles after each unsuccessful attempt. In mathematical terms, th.e timer is set according to the expression

$$T = S\ random\ (0, 2^{min(n,10)})$$

where *random* returns a random number evenly distributed between its two arguments, n is the retry number, S is a proportionality constant, and the min function sets a maximum bound on the retransmission interval. This causes successive attempts by each station participating in a collision (there could be many stations all jamming each other simultaneously) to be spaced out over rapidly increasing intervals until the offered load on the channel drops to the point where successful transmissions can occur. As usual, should some retry limit (typically 16) be reached the packet is abandoned.

Base 2 gives reasonable performance and is easy to implement. However, other values can be used in the exponential calculation. Smaller bases back off less quickly, while larger bases cause rapid increases in the retransmission interval. If retransrnissions are more often caused by poor link margins than by collisions, a smaller base (i.e., slower backoff) may be appropriate at the expense of slower adaptation to congestion.

Packet radio differs from Ethernet in that there is no immediate indication of a collision; one can be detected only by the non-receipt of an ACK. Furthermore, the interval between transmission of a packet and the receipt of an ACK will vary depending on external factors such as channel speeds and the number of digi-

peaters. Therefore, more work is needed to adapt our backoff strategy to packet radio.

4.2. Retransmission Timing

In the most common implementations of AX.25 the estimated interval between transmission of a packet and reception of its ACK is called FRACK and is sent manually. An ad-hoc rule is used to increase it according to how many digipeaters are included in a connection.

There are several problems with this approach.

1. A constant value cannot adapt to changing channel conditions. A value that gives efficient and quick response when the channel is lightly loaded may cause many unnecessary retransmissions when ACKs are slow in returning because of channel load.
2. Few users bother to tune this value to an appropriate value. If anything, users are more likely to set it too small because they get anxious when their TNCs don't transmit when they "should"!

The presence of digipeaters is what makes this problem so difficult. There is no direct way to tell how much channel loading exists in sections of the network out of direct radio range, and hence no way to predict in advance how long one should wait before assuming that a transmission was lost in a collision. When digipeaters are phased out in favor of direct links between adjacent network nodes, a fixed interval with a timer that runs only when the channel appears to be clear should work well. In the meantime, however, we can borrow a technique from TCP and attack the static FRACK problern on long digipeater routes by computing it automatically. To do this we measure the time between the transmission of a given packet and the receipt of its ACK, always making sure that the retransmission timer is greater than this period. A "round trip timer" is created, providing continuous packet-by-packet measurements that take into account changing channel and digipeater loading.

Measured round trip times are only educated guesses about future behavior, because the path is statistical in nature (e.g., someone might unpredictably start a large file transfer through a remote digipeater). Note also that once the round trip timer is started, it should be allowed to run even if the packet being timed is lost and retransmitted. While this can cause anomalously large "spikes" in the round trip time measurements, the alternative (restarting the timer when retransmitting) is worse because an ACK for a previous transmission of the packet might come back immediately after the retransmission. This would yield an erroneously small estimate that might never be corrected. In any event, it is a good idea to process these measurements in two ways before using them to set the retransmission timer.

1. Compute a running average over several measurements to smooth out random fluctuations.
2. Include a "grace" factor to allow for sudden increases in the round trip delay.

The TCP specification recommends this formula for smoothed round trip time computation:

$$T_s' = \alpha T_s + (1-\alpha)\, T$$

where T_s' is the newly computed Smoothed Round Trip Time, T_s is its previous value, T is the round trip time encountered on an ACK just received, and α is a "tuning" constant ranging between 0 and 1. Large values of α cause T_s to react slowly to changes in measured round time, while small values cause it to respond more quickly. The TCP spec recommends an α value between 0.8 and 0.9. Dave Mills, W3HCF has shown [8] that using different values for α depending on whether the delay is increasing or decreasing has merit in the Internet environment. His experiments suggest using $\alpha = 3/4$ when $T > T_s$ and $\alpha = 15/16$ when $T < T_s$ to give a "fast attack" and "slow decay" response characteristic.

Once T_s has been computed, TCP specifies that it be multiplied by another tuning constant, β, to arrive at the final retransmission timer value. The recommended value for β is 2.0, i.e., the transmitter waits two round trip times for an ACK before retransmitting. The TCP specification also recommends that repeated unsuccessful transmissions be spaced out, although the exact backoff algorithm is not specified.

For AX.25 use, we can combine the round trip timer from TCP with the binary exponential backoff algorithm from Ethernet

and use them together to set the retransmission timer. To do this, we first observe that we should always wait at least one round trip interval (preferably β intervals, to allow a grace interval as in TCP) for an ACK to come back. One possible result is the following:

$$T = T_s \beta \, random(1,2^n)$$

When packets are not being lost, the round trip timer will always be set to β times the smoothed round trip interval. Should several retransmissions occur in a row, however, both the average retransmission interval and variance will double on each retry.

4.3. Persistence

The round trip timing and backoff algorithms just described help stabilize the channel and prevent congestion collapse through negative feedback that attempts to operate the channel near the peak of its throughput efficiency curve. These are especially helpful when many hidden terminals are present, limiting the maximum attainable channel efficiency. Even without hidden terminals, however, efficiency can still suffer greatly in a heavily loaded network because of the jump-on problem.

One way to alleviate this is to change how stations with packets to send behave when they find an idle channel. Our current algorithm, namely "transmit as soon as you hear the channel go clear," is formally called *1-persistent CSMA* and leads to the jump-on problem. A less greedy alternative is *p-persistent CSMA,* where each station waits a random amount of time before transmitting even when the channel seems to be clear. Each station generates an evenly distributed random number between 0 and 1 and compares it to a constant, *p,* which also ranges between 0 and 1. If the random number is less than *p,* the station transmits; otherwise it waits a small amount of time (called the *slot time,* ideally one round-trip propagation delay) and repeats the procedure. As *p* approaches zero, channel efficiency theoretically approaches 100%. Unfortunately, however, delay also approaches infinity! For any non-zero value of *p,* however, the packet will eventually be sent after finite delay although channel efficiency will decrease.

Persistence works because the stations waiting for the channel will "spread out" their transmission attempts once the channel goes clear. Assuming only one station decides to transmit in each slot, or round trip interval, the other stations on the channel will hear and defer to it before they attempt to transmit themselves.

Clearly there is an optimum value of *p* for a given level of channel activity. If *p* is too large, too many stations will jump on each other in the first slot. If *p* is too small, many slots will go to waste before the stations eventually transmit, and transmission delays will increase unnecessarily. It is therefore best to set *p* dynamically, if possible. At low load levels, $p=1$ gives the best performance; as the channel reaches 75-80% offered load (i.e., the channel appears busy 75-80% of the time with either good packets or collisions) the value should be decreased.

The backoff algorithm given in the previous section can be viewed as a way to adjust the value of *p* dynamically when timeouts occur, since doubling the retransmission interval corresponds to halving the value of *p.* The only real difference is that the backoff interval is evenly distributed while the persistence "timer" is exponentially distributed.

TAPR TNCs pioneered the use of DWAIT, a persistence-like feature. It should be enhanced to provide true persistence (i.e., by randomization). A worthwhile research project would be the development of a good algorithm for the real-time evaluation of *p*, with the goal of incorporating it into a future revision of the AX.25 specification. One possible approach is to sample the state of the DCD line periodically and estimate the channel activity. The value of *p* could then be found in a lookup table. If more retransmissions occur than expected for the measured amount of traffic (e.g., due to hidden terminals), then *p* could modified as appropriate.

5. Some Final Thoughts on Congestion Control

Congestion control in packet radio networks and packet switching networks in general are notoriously difficult problems and have received much research attention. The suggestions made here, however, should greatly alleviate the problem and make our AX.25 digipeater networks halfway usable under heavy load.

However, it may turn out that radically new approaches to transmitting bits and packets over amateur radio links will hold a better solution.

1. Spread spectrum with different spreading sequences assigned to each receiver eliminates channel collisions except for the less likely case where two transmitters attempt to send to the same receiver at the same time.
2. Since the received signal levels in a packet radio network usually vary widely, modulation methods and forward-error-correcting (FEC) techniques that exhibit higher degrees of "capture" may help by minimizing collisions where nobody "wins."
3. Busy Tone Multiple Access (BTMA) [3] involves receivers indicating on a separate radio channel (with a "busy tone") that they are actively receiving a packet. Stations monitor the busy tone rather than the data channel to determine when to defer. If the power levels and propagation conditions between the main and busy-tone (channels are symmetric, it is possible with BTMA to avoid completely the hidden terminal problem. While this technique requires twice as many radios (along with the ability to operate in full duplex, although this could be crossband) the improvement in throughput could easily more than double.
4. Conventional "bent-pipe" analog or regenerative digital repeaters of the split-frequency type also reveal hidden terminals. While this is contrary to the trend toward digipeaters, they may be the most expedient way to solve a particularly difficult case.

This list only touches the list of possible approaches to congestion control. Above all, we need to leave plenty of room for experimentation into these and other "low level" problems. Our experience has shown that there *cannot* be a single, "standard" link level protocol that is optimum everywhere. Each link protocol must be customized to a specific environment, and the network architecture (e.g., higher level protocols) must take this fact of life into account and be flexible enough to accommodate them all.

6. A Replacement for LAPB

As we have seen, LAPB is out of its natural element (a reliable, full duplex point-to-point link) when it is used on a lossy, half duplex packet radio channel. LAPB lacks the features needed to control congestion and improve performance on bad links. It also contains unnecessary "features" that increase the size of an implementation and may actually degrade performance. This section is an attempt to apply the lessons learned from LAPB to the design of a new protocol, one much better suited to the amateur radio environment.

6.1. Link Level Acks versus Connections

An important point needs to be stressed here. Link level *acknowledgements* (desirable for performance reasons when operating on a lossy channel) do *not* necessarily imply link level *connections.* A packet switch operating in a large metropolitan area may serve a total of several hundred users, but only a small fraction may be active at any one time. Why require the switch to maintain hundreds of mostly idle link level protocol control blocks for all these users, or alternatively saddle users with the burden of setting up a link level "connection" to their local switch before they may communicate with their final destinations'!

Veterans of the amateur packet radio "protocol wars" will recognize this as an argument for datagram-oriented networks. We believe that the connection- (or virtual-circuit-) oriented nature of LAPB is another major contributor to its complexity. We further believe that a redesigned protocol that enhances link level reliability with acknowledgements *without* explicit connection management procedures will be both simpler and easier to implement.

6.2. Window Control

As shown by our analysis of LAPB, sliding windows are unnecessary in a half duplex packet radio environment, so our replacement is a simple stop-and-wait protocol. Even when traffic is pending to more than one station, only one data packet may be sent at a time. An ACK must be received for this packet (or a "give up" interval exceeded) before another packet can be sent to this or any other station. This avoids the ACK collisions that could occur if we were to sent traffic to two or more

stations at one time. This also simplifies the implementation, since all traffic to be sent on a given channel can be kept on a single queue regardless of destination.

6.3. Acknowledgement Piggyback

Another feature of LAPB that makes its contribution to complexity is the ability to "piggyback" an ACK on a data frame traveling in the reverse direction. This is a difficult feature to use in practice, because there is seldom a data frame available at precisely the right time on which to piggyback an ACK. Most applications operate in a "pseudo half duplex" mode: one host sends one or more packets to its peer, and one or more packets later flow in the opposite direction. Some protocol implementations delay the transmission of ACKs in the hope that a higher layer will soon generate an outgoing data frame on which it could be piggybacked. This seldom succeeds, however, because any delay must be kept short both to prevent unnecessary retransmission and to keep the round trip time small enough to avoid affecting throughput. We therefore felt that ACK piggybacking was not necessary in our protocol.

6.4. Acknowledgment Retransmission

In LAPB, a lost ACK causes the sender to timeout and retransmit a data frame, even though it has already been received correctly. When loss rates are low, this is not a serious problem. However, should a substantial fraction of the ACKs be lost, the unnecessary retransmission of data frames by the sender merely to elicit ACK retransmissions from the receiver can cause a considerable loss of performance. This problem was first recognized eleven years ago by the designers of the first packet radio network, the ALOHANET, who proposed an elegant solution that we have dubbed the ACK-ACK protocol[4] [11].

As its name implies, the ACK-ACK protocol provides for the acknowledgement and retransmission, if necessary, of ACKs to prevent the unnecessary retransmission of data packets. Let's look at an example using LAPB more closely to show why ACK-ACK gives better performance. Assume that the probability of a data packet or ACK being successfully received in one try is p_d or p_a, respectively. In order for the sender to expect to receive one ACK, the receiver will, on the average, have to transmit its ACK $\frac{1}{p_a}$ times. In order for the receiver to transmit this many ACKs, however, it will also have to receive $\frac{1}{p_a}$ data packets. Therefore, the sender can expect to send $\frac{1}{p_d} \times \frac{1}{p_a}$ packets for each ACK received. For example, if the probability of a data packet or ACK being successfully received is 25%, then $p_d = p_a = 0.25$, and the sender will have to send each data packet an average of sixteen times before an ACK is finally received and it can continue with the next data packet! On such a link, a greater attempt by the receiver to ensure receipt of its ACKs by the sender is worthwhile.

Given that nothing can be done about the "raw" value of p_a, then the only thing the receiver can do to increase reliability is to retransmit ACKs whenever necessary. If the receiver were to retransmit each ACK up to N times, then the probability that at least one attempt succeeds is

$$1 - (1-p_a)^N$$

If the probability that a data packet is successfully received is p_d, then the expected number of data packet transmissions that result when the receiver makes N attempts to transmit each ACK is

$$\frac{1}{p_d} \times \frac{1}{1-(1-p_a)^N}$$

For our earlier example where $p_d = p_a = 0.25$, if $N=5$ then the probability becomes .76 that at least one ACK will make it back to the sender. This decreases the expected number of data packet transmissions from 16 to 5.25, a substantial improvement.

The receiver does this by setting a timer when it transmits its ACK and retransmitting it if an acknowledgement of its acknowledgement (an "ACK-ACK") does not return before the timer expires. The receiver may also accept the next data packet from the sender as confirmation that its ACK of the preceding one was accepted, because the sender will not continue with the next data packet until the one

[4] We considered calling this the "Bill the Cat Protocol," but thought the reference too obscure.

outstanding has been acknowledged (remember this protocol operates in stop-and-wait mode). This means that when one station sends a steady stream of traffic to another, no additional packets over the conventional protocol case are sent. Only at the end of a burst of traffic (or after an isolated packet) is an extra ACK-ACK packet generated. Clearly, the ACK retransmission timer must be shorter than the data retransmission timer; this ratio corresponds to the value of N in the above equations, the number of times the receiver will attempt to retransmit the ACK before the data packet is retransmitted unnecessarily.

To make the protocol reliable, we need to include a *frame identifier* (ID) with each transmitted 'data frame and ACK. The ID ensures that the sender and receiver do not get out of step, possibly losing or duplicating a data frame. The ID need not be a sequential value; it need only be different from one frame to the next. If the receiver receives a data frame with the same source address and ID as the last received frame, a normal ACK is sent but the frame is otherwise ignored as a duplicate.

The ACK-ACK protocol can be viewed as establishing an implicit, unidirectional "connection" with the first data frame, which exists only as long as there is data to send. Once an ACK-ACK is sent to show that no more data is available, the sender and receiver need retain no further state (i.e., addresses and ID) and the "connection" is torn down. The sender then repeats the: process with any other station for which it has traffic.

Only one implicit "connection" exists at any moment (although the destination to which we are "connected" can change very rapidly) since our half duplex operating rules require that only one data packet be outstanding at a time. If a third station sends us data while we're already exchanging data with another station, then this new data is put on a queue and processed after we've completed our current transfer sequence. If a data frame requesting an ACK should arrive from a new station, we do not immediately acknowledge it because that would encourage it to send more data. This might interfere with the station we're already communicating with, so we hold this frame on a queue for processing once we return to an idle state. (However, an ACK from such a station may be processed immediately, as may incoming data that does not request an ACK). There: is of course the chance that we might delay processing of the new data so long that a tirneout occurs. This can be largely avoided, however, if the link retransmission timers are chosen to be longer than the time needed to send a typical multi-packet "burst." These can be limited by appropriate window sizes in the end-to-end (transport) protocol. The alternative, immediately acknowledging the new data but telling the sender to hold off on more, is possible but complicates the protocol. Periodic "probes" from the other station would be needed to guard against the deadlock that would occur if our "go ahead" message is lost, and these could also collide with packets to or from the station we're already communicating with. The simple acknowledgement delay strategy thus seems workable, especially if the data retransmission strategy backs off rapidly (e.g., exponentially). It also simplifies the code greatly, since managing multiple connection state control blocks is usually the hardest part of a "multi connect" TNC.

Since radio channels can vary widely in quality, it is desirable to make the use of ACK-ACKs and even ordinary ACKs optional. This is especially useful when supporting datagram-oriented network layer protocols such as IP, since it is more efficient to dispense entirely with the overhead of link level ACKs and rely on end-to-end retransmission of lost packets when link error rates are very low. Our protocol therefore provides for an indication in each packet (data or ACK) whether a reply (ACK or ACK-ACK) is expected for this transmission. It is therefore easy to adapt this protocol to changing radio conditions or user reliability requirements. An ACK-ACK need be sent only when no more data is available, since the next data packet awaiting transmission may serve as an ACK-ACK. An ACK-ACK may then be represented merely by an empty data packet with the "acknowledgement requested" bit turned off. This simplifies both the concept and the implementation of the protocol considerably.

6.5. ACK-ACK Preliminary Specification

In specifying frame formats corresponding to data, ACK and ACK-ACK messages, we have tried to adhere to the basic structure of the AX.25 datagram sublayer. In other words, we keep the standard AX.25 address field lay-

outs and attempt to use the existing control flag definitions whenever possible. Data is transmitted as a UI frame with poll and command turned on, ACK is transmitted as a UA frame with final and response turned on, and ACK-ACK is transmitted as an empty (no data in the data field) UI frame with poll turned off and command turned on.

6.5.1. Frame Formats

The frame format is similar to that of AX.25. The fields are named and the length of the field in bits follows the field name and is enclosed in parenthesis. Here is the basic frame:

F (8)	Dest (56)	Src (56)	Digi *(O-448)*	Ctl (8)	PID (8)	ID (8)	Data	FCS (16)	F (8)

Where

F Flag, length *8* bits. Value 7E hex, not bit stuffed.

Dest Address of destination, length 56 bits. This is the standard AX.25 address/SSID combination.

Src Address of source, length 56 bits. This is the standard AX.25 address/SSID combination.

Digi Address of repeater(s), length varies from 0 (no digipeaters) to 448 (8 digipeaters). This is carried over from AX.25, although deprecated for the networks in which this protocol is likely to be used.

Ctl Control field, length *8* bits. The content determines the frame type, either UI or UA. The following values are in binary:

UI 000p0011

U A 011p0011

where 'p' is the poll/final bit.

PID Protocol identifier, length 8 bits. This identifies the layer three protocol and follows the same conventions as AX.25.

ID Frame ID, length 8 bits. This uniquely identifies the frame from the previous and subsequent frames to avoid duplicate and lost frames.

Data Data, length 0 to 523,688 bits (0 to 65,461 bytes). The sender and receiver should agree on a maximum frame length not to exceed 65,461 bytes in length.

FCS Frame check sequence, length 16 bits. This contains the CRC-CCITT checksum of the frame.

F Trailing flag.

The ID byte is probably best set from a single counter used for all transmissions regardless of destination. The only requirement placed on the sender is that the value of the ID field not be duplicated in any two subsequent frames sent to the same destination; this can be avoided by limiting the transmit queue to 255 entries.

6.5.2. Flow of Data Between Stations

This section is a preliminary specification of the ACK-ACK protocol in an informal, narrative style. As this definition is refined and implemented, a more formal description is being written. Contact the authors for more information.

Data is transferred from station A to B in the following way. Station A selects a ID value, sends the data (UI) frame to B, and starts a timer with interval T_d. B notes the sender address and ID number, responds with an ACK (UA) containing the ID just received, and starts a timer with interval T_a. After receiving an ACK. from B for the most recently sent frame, A responds by sending the next data frame and restarting its timer if there is more data to send, or sending an ACK-ACK (empty data frame without poll request) and stopping its timer if there is no more data to send. When B receives the next data frame or an ACK-ACK, B discards any ID it is currently retaining from A and either sends an ACK with the new ID (restarting its timer) or goes into the idle state (stopping its timer) as appropriate.

If A should fail to receive an ACK containing the correct ID from B within the timeout interval T_d, then it *increments* a retry counter, retransmits the: data frame and restarts its timer. If the retry counter exceeds a fixed limit, additional transmission attempts for this frame are aborted, and if possible, the packet should be returned to its original source with an error indication. Higher level protocols are then responsible for any further error recovery procedures. If B fails to receive either another data frame or an ACK-ACK from A within its timeout period T_a, then it resends its ACK,

restarts its timer and increments a retry counter. If the same data frame is received again from A, B resets its retry counter, resends an ACK and restarts its timer but otherwise ignores it as a duplicate. If there is no response at all from A after N ACK retransmissions, where N is the ratio $\frac{T_d}{T_a}$, B abandons any further retransmission at tempts and acts as though an ACK-ACK had been received (i.e., it discards the ID and sender information and returns to a quiescent state). Higher level protocols are responsible for detecting and recovering from the residual possibility for packet duplication this allows.

Because of its small amount of state, this protocol can be implemented simply and with little memory. Communicating with several different stations in "rapid fire" sequence is easy; the receiver need retain the last ID valued received from a particular sender only as long as data is actively being transferred. Special connection establishment packets are unnecessary.

7. Selection of Appropriate Values for Tuning Parameters

There are three parameters that should be regularly adjusted or "tuned" to the existing conditions on the channel. These parameters are frame size, transmit timeout timer T_d, and ACK timeout timer T_a. The frame size should be adjusted to the optimum value based on the current channel statistics. As more frames are lost the frame size should be shortened until the optimum operating point is reached as described in Appendix 1.

The correct timer values are much simpler to calculate. The transmit timer should be set to some value that is significantly greater than the round-trip time for a frame to travel from sender to receiver for an ACK to return. The receiver should also calculate the round-trip time and determine how many ACKs are required to insure that an ACK reaches the sender. The receiver should set the value of T_a to be T_d divided by the number of ACK transmissions N necessary to increase the probability that an ACK arrives at the sender to the desired level. In order for this technique to work properly both the sender and receiver need to agree on the formula that relates round trip time to the selected value of T_d. Both timers should be randomized and backed off as described earlier in the congestion control section.

As soon as T_d (equal to NT_a) expires at the receiver the receiver should cease sending ACKs. This prevent the ACKs from colliding with and destroying a retransmitted data frame. If the sender should fail to receive an ACK it will resend the frame. The receiver repeats the process of acknowledgement but discards the frame.

7.1. Performance

ACK-ACK was compared to AX.25 for various links and message sizes using a Monte Carlo program that simulates both ACK-ACK and AX.25 in identical environments. In all configurations a window size of 1 (stop and wait) was found to be most efficient and ACK-ACK to be more efficient than LAPB/AX.25. Performance improvements over LAPB/AX.25 ranged from a low of 1.21 for very reliable links to a situation involving 2 digipeaters where ACK-ACK delivered the data and AX.25 failed to deliver all the data within the running time of the simulation. In the latter case, when the simulation was terminated the performance improvement over AX.25 was already over 100.

7.2. Conclusion

LAPB/AX.25 is a good protocol for point-to-point communication over reliable full-duplex links. ACK-ACK is more desirable than AX.25 in high error rate and/or half-duplex environments because it provides significantly greater throughput, better channel utilization, and is much simpler to implement.

8. References

[1] Fox, T., ed "AX.25 Amateur Radio Link Layer Protocol Version 2'', American Radio Relay League, 1985.

[2] CCITT Study Group VII, "Interface Between Data Terminal Equipment (DTE) and Data Circuit-Terminating Equipment (DCE) For Terminals Operating in the Packet Mode on Public Data Networks," Recommendation *X.25, 1984.*

[3] Kleinrock, L., and Tobagi, F., "Random Access Techniques for Data Transmission over Packet-Switched Radio Channels," National Computer Conference, 1975. pp187-201.

[4] Gower, N., and Jubin, J., "Congestion Control Using Pacing in a Packet Radio Network," IEEE Conference on Military Communications, 1982.

[5] Metcalfe, R. M., and Boggs, D. R., "Ethernet: Distributed Packet Switching for Local Computer Networks," Commun. ACM, vol. 19, pp. 395-404, July 1976.

[6] Postel, J., ed., "Transmission Control Protocol Specification," ARPA RFC 793, September 1981.

[7] US Department of Defense, "Military Standard Transmission Control Protocol," MIL-STD-1778, 12 August 1983.

[8] Mills, D., "Internet Delay Experiments," ARPA RFC 889.

[9] Brady, Paul T., "Performance Evaluation of Multi-Reject and Selective Reject Link-Level Protocol Enhancements," Bell Communications Research. (Paper submitted to ICC Toronto, June 1986).

[10] Nagle, J., "Congestion Control in IP/TCP Internetworks," ARPA RFC 896

[1 1] Binder, R., et al., "ALOHA Packet Broadcasting - A Retrospect," Proceedings of the 1975 National Computer Conference, page 208-209.

Appendix 1: Efficiency of a Go-Back-N Protocol

Two interrelated factors must be taken into consideration when evaluating the efficiency of a go-back-N sliding window protocol on a noisy channel: the effect of header overhead (limiting efficiency even on an error-free channel) and the effect of garbled frames. This section derives the formulas for these two factors that were used to arrive at the conclusion that a window size of 1 (i.e., a stop-and-wait protocol) maximizes channel efficiency when operated on a half duplex channel. To determine the net efficiency of the channel, these two factors are evaluated with the desired parameter values and then multiplied.

Header Overhead

If there are D data bits in a transmission, H frame header bits, and N frames in a transmission, then the ratio of data bits to total bits sent is

$$\frac{D}{D+NH}$$

If we include the overhead of an ACK (A bits) then this becomes

$$\frac{D}{D+NH+A}$$

Unusable Frames

If the channel bit errors occur independently at rate R, (i.e., caused by Gaussian sources such as "white" thermal noise) then the probability that a frame of length $D+H$ is received without errors is simply $(1-R)^{D+H}$. In this section we call this quantity G, the probability of receiving a good frame. Note that G decreases as we increase D; i.e., the longer the frame, the greater the chance it will be corrupted in transmission, unless the channel has a perfect bit error rate ($R=0$).

In the go-back-N error recovery technique, only the next expected frame can be processed; any other frames received are discarded, even if they are received correctly (i.e., with a valid CRC).

Consider this example. Four frames numbered 1, 2, 3 and 4 are sent in one transmission. Frame 1 is received normally, but frame number 2 is corrupted. Even if frames 3 and 4 are received correctly, the receiver will only acknowledge frame number 1. Frames 2, 3 and 4 will have to be resent, even though only frame 2 was actually lost.

To analyze the performance of a sliding-window protocol with go-back-N recovery, we need to find the expected number of *usable* (with correct CRC and in correct sequence) frames in a transmission containing N frames. We then divide this number by N, yielding the normalized efficiency of the protocol (the number of usable frames received divided by the total number sent). G is defined as the probability of any specific frame within the transmission being received correctly, and we assume that this value is the same for all frames (i.e., the frames are all the same size and errors are independent). Clearly, if we

were able to use every correctly received frame (i.e., if we did not discard frames received out-of-sequence due to the loss of an earlier frame) then this efficiency would simply be G, for any value of N. In the go-back-N case, however, the analysis is more complicated.

The probability that zero frames are usable is (l-G), i.e., the probability that the first frame is in error, rendering it and all later frames unusable. The probability that only one frame is usable is G(l-G), the probability that the first frame is received correctly and the second one in error, and so on up to the probability that all N frames are usable. Summing over all possibilities and weighting by the: number of usable frames for each, we obtain

$$(1-G)\sum_{i=0}^{N-1} iG^i + NG^N$$

By factoring out G from the series and then integrating and differentiating the finite sum, we obtain

$$(1\text{-}G)\ G\frac{d}{dG}\int\sum_{i=0}^{N-1} iG^{i-1}\,dG + NG^N$$

Integrating,

$$(1-G)\,G\frac{d}{dG}\sum_{i=0}^{N-1} G^i + NG^N$$

This now contains the sum of a finite geometric series. Substituting with the closed form of the sum, we obtain

$$(1-G)\,G\frac{d}{dG}\frac{G^N-1}{G-1} + NG^N$$

Differentiating,

$$(1-G)\,G\left[\frac{1-G^N}{(1-G)^2}-\frac{NG^{N-1}}{1\ -\ \mathrm{G}}\right] + NG^N$$

Simplifying,

$$G\frac{1-G^N}{1-G}$$

And normalizing by N, the number of frames sent in each transmission, we finally obtain

$$G\frac{1-G^N}{N(1-G)}$$

A quick check shows that substituting N=l for the single-frame case yields G, as expected.