Toward New Link-Layer Protocols for Amateur Packet Radio

Phil Karn, KA9Q
karn@unix.ka9q.ampr.org

AX.25

* Created in 1982 - over a decade ago

* Based on CCITT X.25 LAPB (itself based on HDLC and SDLC)

* Defines datagram-style addressing header for use with LAPB

* No allowance for the special problems of radio - collisions, noisy radio links, congestion, hidden terminals, etc

AX.25 - 11 years later

*** What do we have now that we didn't have in 1982?**

***Far* more powerful computers (100-1000x CPU, RAM, disk)**

A variety of RF modems (sort of)

A variety of upper layer protocols (TCP/IP, ROSE, NET/ROM, TexNet)

A *lot* of users

A decade of experience!

AX.25 - Lessons Learned

* Link protocols designed for wired networks (especially by PTT monopolies) don't transition very well to radio

* Efficient channel access is a serious problem in real networks

* Pure ARQ doesn't work very well, especially with QRM like 70cm military radar

* What a link protocol doesn't do is just as important as what it does do (e.g., "connections" are superfluous at the link layer)

* We actually need a variety of link protocols, each suited to a particular channel (wire, HF, VHF/UHF, satellite), and tied together into a uniform Internetwork, e.g., with TCP/IP. CLOVER is one such protocol, specifically designed for HF. We need more.

Forward Error Correction (FEC)

Send redundant information to allow correction of (some) errors without retransmission

Exploit Shannon's tradeoff between bandwidth and S/N ratio:

$$C = B \log_2(1+S/N)$$

The *only* practical way to deal with certain types of interference, e.g., radar

Why Use FEC?

* ARQ discards "bad" packets, while FEC uses *all* of the received energy to produce a (hopefully) error-free packet

* As long as ARQ retransmissions are rare, the loss is minimal
  Rule of thumb: no more than 1% packet loss rate

* We are far worse than this on most amateur links!

* Small packets may help, but adds considerable overhead

* The alternative (increasing power and/or antenna gain) is not always practical or economical

* Brains over brawn!

Why Not Use FEC?

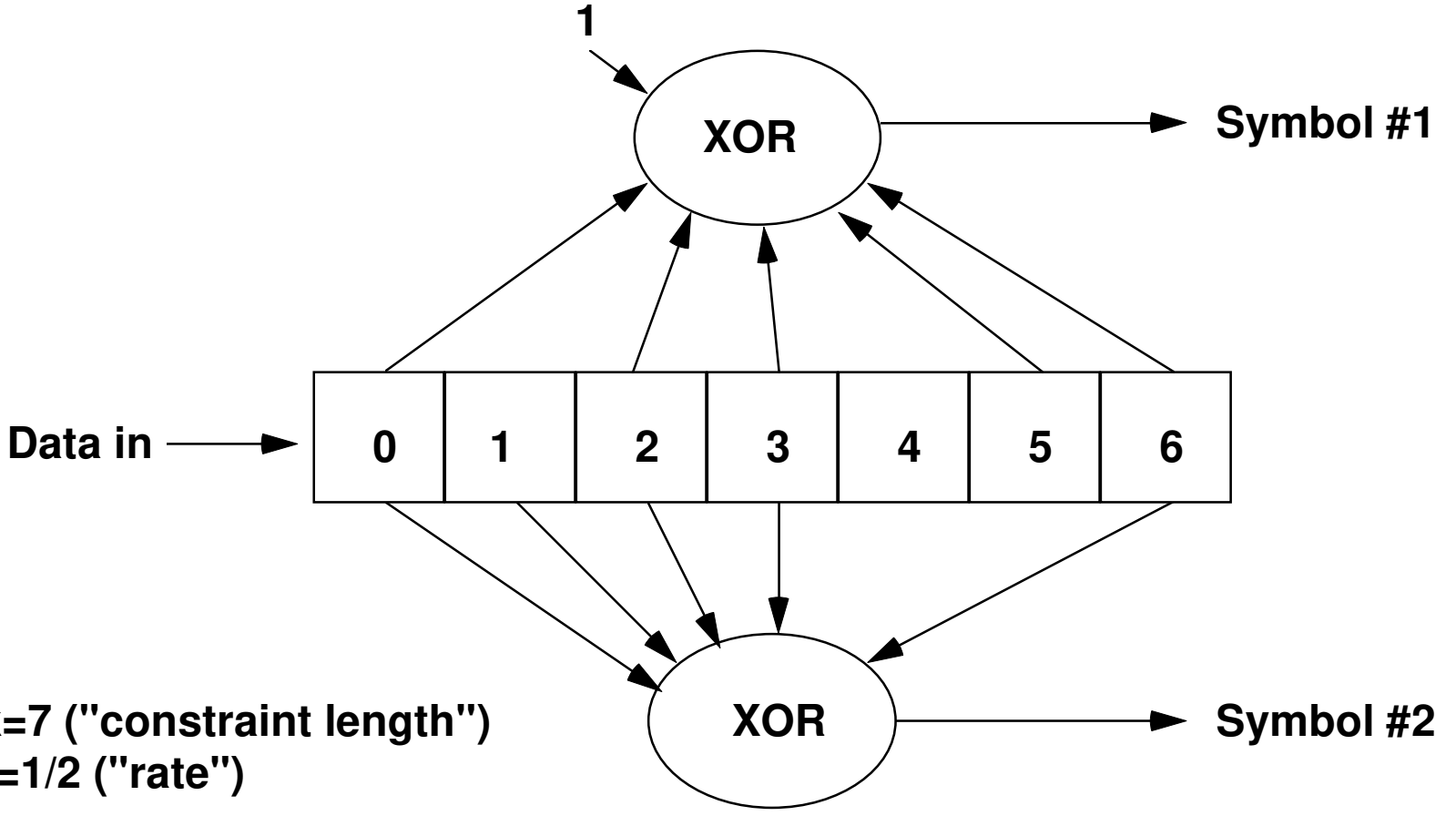
* Constant overhead, whether needed or not

* Compute-intensive to decode

FEC - Some Examples

* Block Coding
    - BCH - AMPS (FM cellular phone) control messages
    - Reed-Solomon (compact disks)
    - Hamming (Microsat computer memories)
* Convolutional (stream)
    - CDMA digital cellular
    - Space communications (VSAT, interplanetary, etc)
    - V.32/V.32bis telephone modems ("trellis coding")

Sample Convolutional Coder



k=7 ("constraint length")
r=1/2 ("rate")

NASA standard
planetary code

Decoding Convolutional Codes

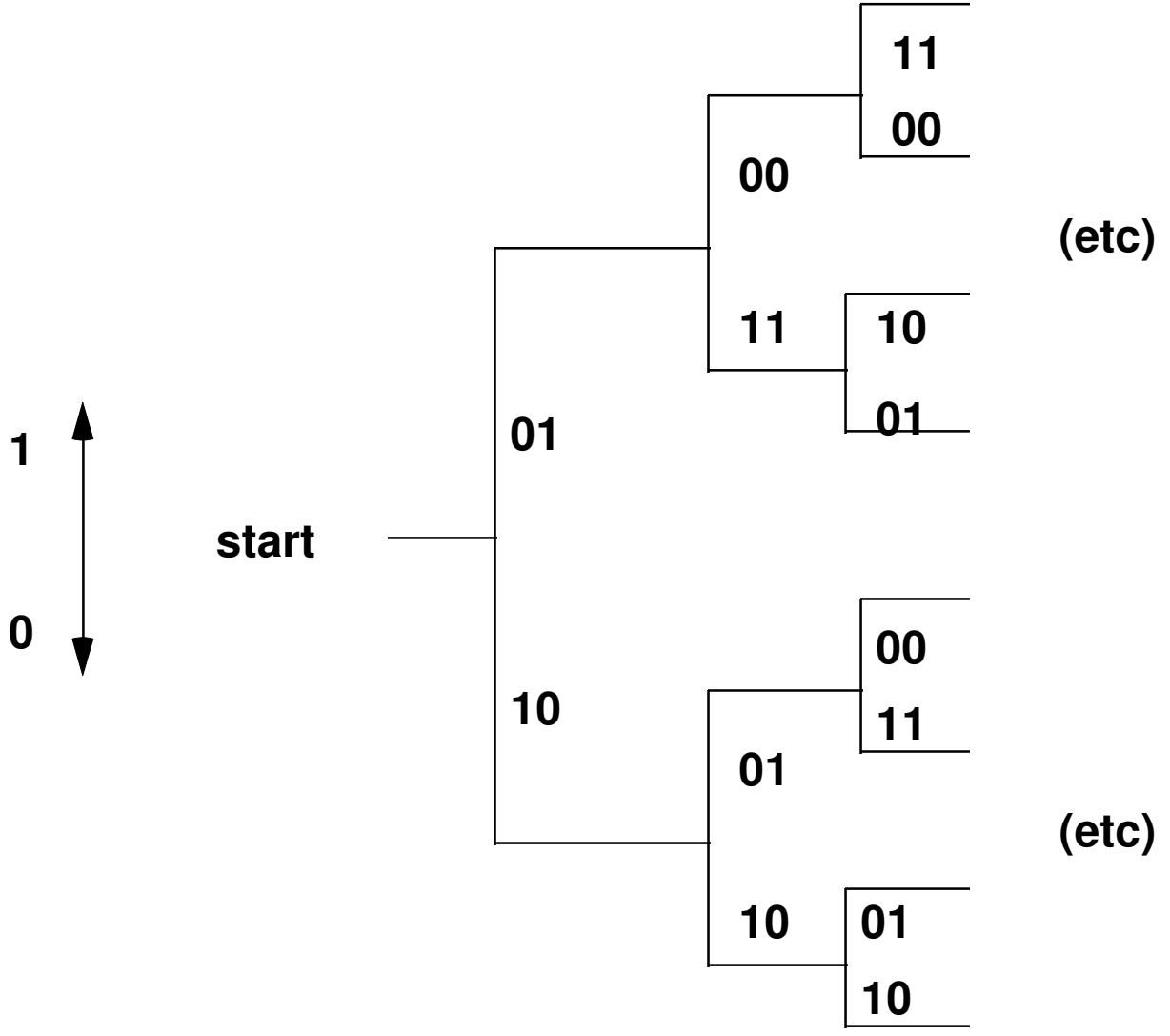
* Parallel (Viterbi)

  Constant decode time
  Limited constraint length - complexity O(2^k)
  Well suited for hardware implementation - chips commercially available
  Best for channels limited by thermal or thermal-like noise, e.g., satellite or spread spectrum

* Sequential (Fano, Stack)

  Variable decoding time, depends on error rate; either faster or slower than Viterbi
  Allows very long constraint lengths (k=32 common)
  With k large, nil probability of incorrect decode (timeout more likely)
  Good for software implementation, especially when error rate is low

Sequential Decoding



Sequential Decoding, contd

* Decoder has a copy of the encoder, compares received symbols with versions regenerated locally

* By "intelligent" trial and error, the decoder finds the data sequence that produces the best match between the locally regenerated symbols and the actual received sequence

* The decoder also produces a "metric", its estimate of the quality of the incoming symbol stream. Like an S-meter, only more useful

* Decoder is very fast when there are few errors because it doesn't have to back up very much; slows down considerably when error rate is very high

A Fano Decoder in C

* "Hard decision" Fano Sequential Decoder in C

* Choice of several r=1/2, k=32 polynomials (e.g., NASA standard)

* On a 33 Mhz 486, decodes 56 kilosymbols (28 kilobits) per second on average with input symbol error rate < 2%

* Proves feasibility of sequential decoding with modern amateur resources

Frame Synchronization

* Need to reliably detect the beginning of a packet in the presence of errors as high as 10% to provide margin for FEC

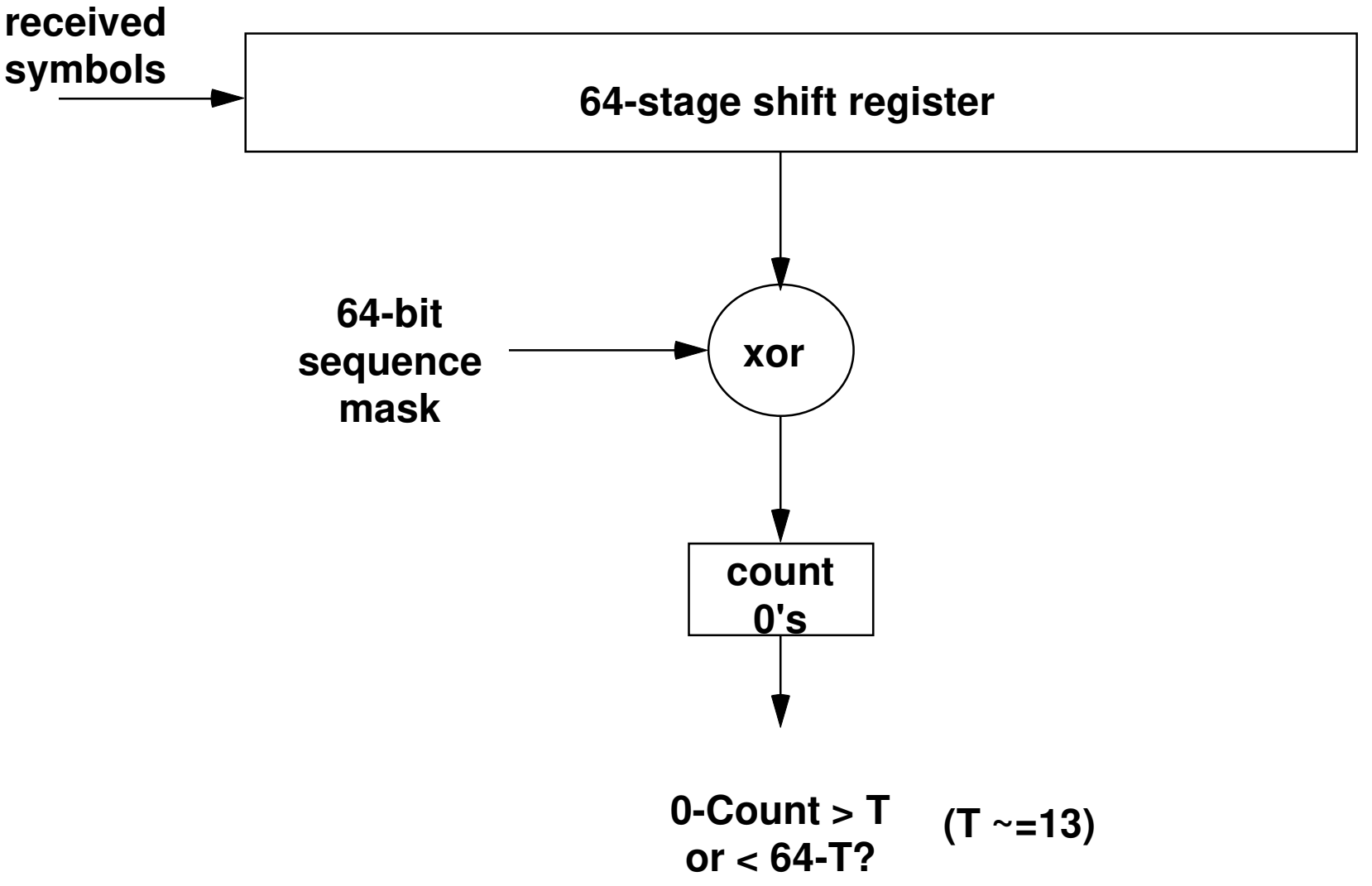
* 8-bit HDLC "flag" unsuitable due to short length

* Use pseudorandom sequence with good autocorrelation properties, e.g., a PN sequence

* The longer the PN sequence, the greater the margin between missing frame sync and falsely detecting one in noise. 64 looks good, and is easy to implement in software on a 32-bit CPU

* Reliable sync enables use of negative acknowledgements (NAKs) to request additional information to aid decoding

Detecting Sync



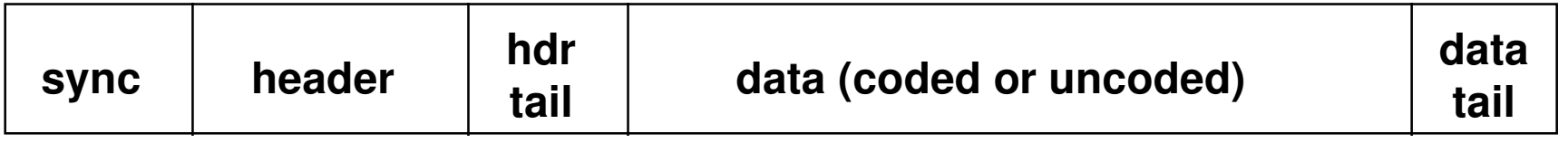
Hidden Terminals

* Proposed solution: Use Multiple Access with Collision Avoidance (MACA), ARRL Computer Networking Conference 1990

* Synopsis: Use RTS/CTS (Request-to-Send/Clear-to-Send) handshake to hold off hidden terminals before actually sending data

* Extra overhead of RTS/CTS handshake can be mitigated with large data packets, which use of FEC facilitates

One Possible Frame Layout



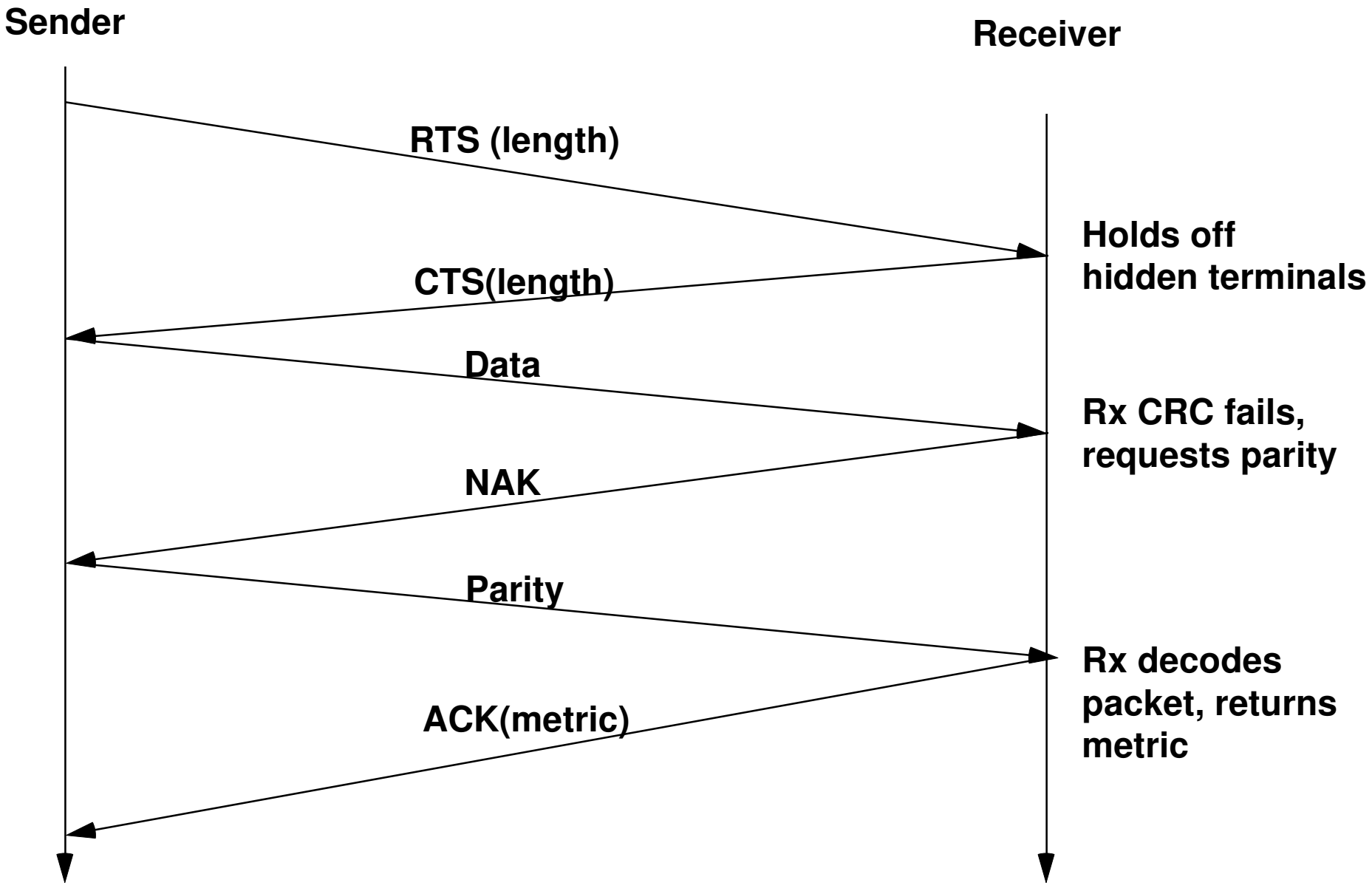
tail replaced with CRC in uncoded data packets

Header contents (always coded):

Source address (callsign)
Destination address
Packet type (RTS, CTS, data, ACK, NAK)
Data length
Data format (data only, parity only, both, neither, interleave on/off)
Decoder metric from last packet received

Use r=1/2 systematic code in adaptive ARQ/FEC hybrid to allow transmission of FEC parity symbols only when needed

Sample Protocol Sequence



Conclusion

Although forward error correction (FEC) has been around in the commercial, military and scientific worlds for some time, only recently have dramatic advances in personal computers brought these powerful techniques within the means of the average radio amateur.

Now is the time to experiment with FEC with an eye toward a whole new generation of amateur packet radio link layer protocols.