# Solving Capture in Switched Two-Node Ethernets by Changing Only One Node


Wayne Hayes
University of Toronto

Mart L. Molle
University of California, Riverside

## Background: History of Ethernet

- designed when 10 Mb/s was enough bandwidth to handle hundreds of hosts

- all hosts share one broadcast medium
  - **collisions** can occur during transmission

  $\Longrightarrow$ need a way to resolve collisions

- *binary exponential backoff* resolves collisions between $N$ hosts in time $\log N$ on average

- if $N = 2$, successive collisions can trick a host into backing off for too long, causing long delays and short-term unfairness

# The Standard Ethernet Protocol

When there are packets to send, locally execute the following:

1. set $attempts := 0$. Remember this for later; the $attempts$ counter is reset for *every* new packet.

2. wait for silence on the network...

3. Attempt transmission! If successful, wait a few bit-times and then go get another packet. Otherwise, a collision has occured...

4. increment $attempts$ by 1

5. choose uniform random integer $delay$ between 0 and $2^{attempts} - 1$ inclusive.

6. Sleep for $delay$ slot-times, where a slot is the time taken to transmit 512 bits.

7. Proceed to step 2.

# Why Ethernet breaks with two busy hosts: The "Capture" Effect.

- Say Alice and Bob each have lots of packets.

- After $k$ collisions, Alice wins and sends a packet while Bob goes back to sleep. Alice continues to send packets.

- Bob wakes up while Alice is still sending packets. They collide after Bob sees the end of Alice's $p$th packet.

- Alice chooses a delay between 0 and 1. Bob chooses a delay between 0 and $2^k - 1$. Guess who wins most of the time?

The Capture effect ... con't

- Alice wins, Bob goes to $k+1$ collisions, and goes to sleep again

- Bob's odds of winning decrease exponentially with each collision

- When Alice runs out of packets, Bob's $attempts$ counter is high and he's sleeping for a long time.

  - so the network is completely idle until he wakes up and starts sending.

  - If Alice gets any new packets, they will get sent even before Bob gets a chance to send his first packet.

Net effects: (no pun intended, of course)

- large "run lengths" resulting in large variance of delay (diagram)

- lots of time wasted idling after a run

This is BAAAAAD

- real-time stuff likes short, predictable delay. (*eg.* realtime audio, video, distributed computing, even just remote typing.)

- long delays confuse higher level protocols (and users!) into thinking something is wrong, when nothing is.

Some really basic ideas:

- Modify one node, the other runs standard Ethernet

- The standard Ethernet node normally hogs the network as long as it wants

- The modified node stops this using the electronic equivalent of a baseball bat

- When the modified node is transmitting, it can keep track of how many times the standard node has collided, *ie.* keep track of the standard node's collision counter.

- When Modified node's turn is over, let the standard node transmit unhindered for awhile.

# SHEP

## Switched Half-duplex Ethernet Protocol

(slightly simplified)

**S** = Standard Ethernet Node

The modified node runs the following:

0. note the wall-clock time at the beginning of my turn.
1. When there is a packet to transmit, wait-for-silence plus the interframe gap, then attempt transmission as in regular Ethernet.
2. If a collision occurs, always choose a backoff delay of 0...
3. ... until **S**'s collision counter reaches a fixed maximum $maxCC$, after an interval of time $T$.
4. The end of my turn has arrived. Concede control to **S**. Be absolutely silent and let **S** transmit unhindered for an interval of time $T$.
5. Proceed to step 0.

Note turn length is stochastic, $maxCC$ is fixed.

- $maxCC = 1$ seems to be the best choice, because letting it get bigger increases delays and increases idle time between switch overs. Exception: long networks – a bit bigger, maybe 2 or 3.

- When to concede? Some choices:

  1. Immediately when **S**'s collision counter reaches $maxCC$

  2. Retry once more.

     $*$ use the idle time that's likely to occur immediately after this collision.

     $*$ strictly bounded delay

  3. "Transmit Damnit!" $\Longleftarrow$

     $*$ "almost" bounded delay, slightly higher capacity.

Comparison to CABEB (& others)

- We change only one node, CABEB (BLAM, full-duplex Ethernet) both nodes need to be changed to get good results.

- Overload with CABEB against standard node $\Rightarrow$ CABEB node hogs bandwidth, even if standard node offers higher load. [Tables 6.1-6.3 of Ramakrishnan & Yang]

- Since CABEB is one-packet-per-turn, packet sizes differ $s1, s2 \Rightarrow$ relative load forced to $\frac{s1}{s2}$ during periods of overload.

- However, SHEP has slightly less capacity for small packets, since extra collisions & idle time between turns.

## Summary, Conclusions

With two busy nodes running standard Ethernet protocol, one node can "capture" the network, causing

- significant short-term unfairness
- frequent large delays

SHEP

- efficient, round-robin service
- decreases std. dev. of delay by 2 orders of magnitude
- eliminates large delays
- negligible cost in bandwidth