10 Gigabit Ethernet

The New Frontier

Presentation by Rupert Dance

Introduction

- Ethernet Legacy
- New Features
- Implementation
- Conclusion

The Legacy of 10 Gb Ethernet

- Frame size
  - Minimum frame size 64 bytes
  - Maximum untagged frame size 1518 bytes
- Half duplex
  - Supported from 1 Mb through Gigabit
  - No half duplex in 10 Gb Ethernet
- Full duplex - no CSMA/CD
  - Simultaneous transmission and reception
  - No carrier sense and no collision detection
  - No frame extension and no frame bursting

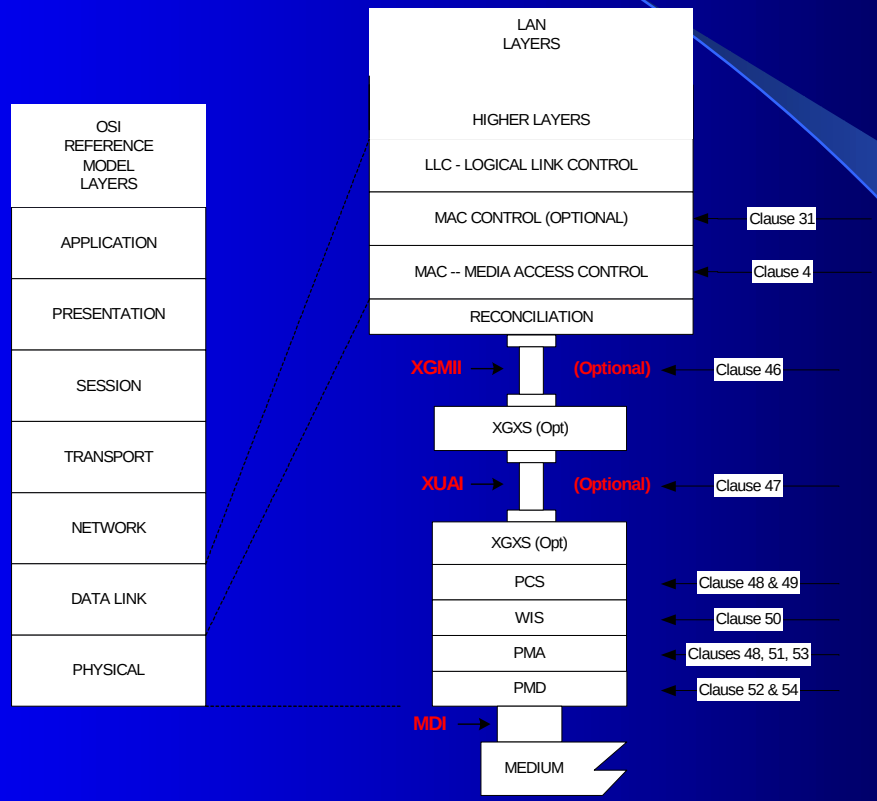
Comparisons
Maintaining Backwards Compatibility

Parameters	**1 Base-5**	**1 Gb/s**	**10 Gb/s**
	10 Mb/s		
	100 Mb/s		
slotTime	512 bit times	4096 bit times	not applicable
interFrameGap	96 bits	96 bits	96 bits
attemptLimit	16	16	not applicable
backoffLimit	10	10	not applicable
jamSize	32 bits	32 bits	not applicable
maxUntaggedFrameSize	1518 octets	1518 octets	1518 octets
minFrameSize	512 bits (64 octets)	512 bits (64 octets)	512 bits (64 octets)
burstLimit	not applicable	65 536 bits	not applicable
ifsStretchRatio	**not applicable**	**not applicable**	**104 bits**

HSSG Objectives - Nov 1999

- Support a speed of 10.0 Gb/ s at the MAC/ PLS service interface
- Define two families of PHYs:
  - A LAN PHY, operating at a data rate of 10.0 Gb/ s
  - A WAN PHY, operating at a data rate compatible with the payload rate of OC- 192c/ SDH VC- 4-64c
- Define a mechanism to adapt the MAC/ PLS data rate to the data rate of the WAN PHY

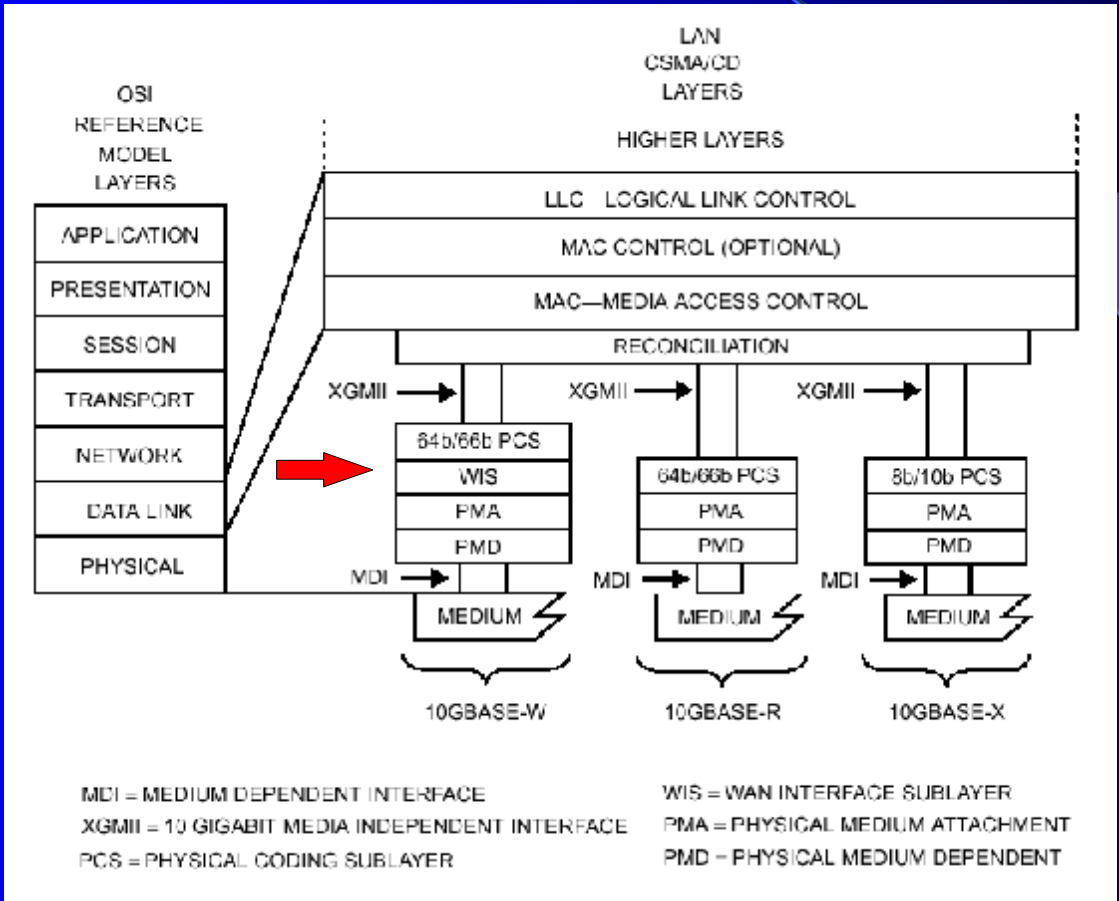
OSI Model – 10 Gigabit



MDI = Medium Dependent Interface
PCS = Physical Coding Sublayer
PMA = Physical Medium Attachment
PMD = Physical Medium Dependent

WIS = WAN Interface Sublayer
XAUI = 10 Gigabit Attachment Unit Interface
XGMII = 10 Gigabit Media Independent Interface
XGXS = XGMII Extender Sublayer

Rate Control



MAC - PHY
Rate Control Alternatives

- Fine granularity rate control
  - Word- by- Word
- Packet granularity rate control
  - Carrier Sense based
  - Busy Idle
  - Self pacing in the MAC

Rate Control Alternatives
Word-by- Word

- Adds a “hold” signal on the XGMII from the PHY to the MAC
  - MAC stops transmission for one clock cycle
  - The MAC inserts “nulls” into the data stream
- Issues
  - Interrupts the flow of data through pipeline stages
  - Makes buffer pre- fetching difficult
  - Tricky timing
  - MAC is no longer a scaled version

Rate Control Alternatives
Busy Idle

- PHY sends “Busy Idle” to MAC during IPG
  - MAC pauses transmission at frame boundary
- PHY sends “Normal Idle” to MAC during IPG
  - MAC resumes transmission
- Need a ~256 byte FIFO in WAN PHY Tx path
- Issues
  - Too much overhead
  - PHY has to monitor FIFO and send blocking signal
  - MAC has to monitor signal and block transmission

Rate Control Alternatives
MAC Self-Pacing

- MAC “knows” the PHY is slower and by how much
- MAC adapts its average data rate by extending the IPG after each frame transmission
- MAC never exceeds the average data rate in the PHY, with packet granularity
- IPG extension is “dynamic” --- depends on the size of the last transmitted frame
- PHY is only required to sustain the transmission of one maximum size packet
- Requires a rate adaptation FIFO in the PHY of ~64 bytes (plus framer overhead)

MAC Self-Pacing
Pascal Variables - 4.2.7.2

- ifsStretchMode: Boolean
  - Indicates the desired mode of operation,
  - Value does not change between invocations of the Initialize procedure
- ifsStretchCount: 0..(ifsStretchRatio – 1)
  - Counts the number of bits during a frame's transmission that are to be considered for the minimum interFrameSpacing extension
- ifsStretchSize: 0..(((maxUntaggedFrameSize + qTagPrefixSize) x 8 + headerSize + interFrameSpacing + ifsStretchRatio – 1) / ifsStretchRatio);
  - Counts the integer number of octets that are to be added to the minimum interFrameSpacing
- ifsStretchRatio
  - Determines the number of bits in a frame that require one octet of interFrameSpacing extension
- Why is the magic number 104?
  - Because 104/112 => .9285714 - approximately the WAN Phy rate

Deference Process

```
while (realTimeCounter > 0) do {Time out entire interframe gap}
   begin
      if ifsStretchMode then {Adjust for minimum IFS transmission}
         begin
            ifsStretchCount := ifsStretchCount + 1; {Count the bits during minimum IFS}
            if (ifsStretchCount = ifsStretchRatio) then {Reached the "magic" number}
               begin {Extend the IFS by one more octet and clear the bit-count}
                  ifsStretchSize := ifsStretchSize + 1;
                  ifsStretchCount := 0
               end
         end
      else nothing;
      Wait(1);
      realTimeCounter := realTimeCounter – 1
   end;
if ifsStretchMode then
   begin
      while (ifsStretchSize > 0) do {Extend the minimum IFS}
         begin
            Wait(8);
            ifsStretchSize := ifsStretchSize – 1
         end;
      if not frameWaiting then {Don't roll over the remainder into next frame}
         begin
            Wait(8);
            ifsStretchCount := 0
         end
   end;
```

Deference Process - How it works

- Rate control occurs in the deference process
- After the completion of timing the interFrameSpacing, the Deference process continues to enforce interframe spacing for an additional number of bit-times
- This is determined by the Bit-Transmitter process, and is reflected in the variable ifsStretchSize.
- If the ifsStretchCount contains a non-integer bit count, and the next frame is ready for transmission, the process enforces interframe spacing for only the integer number of octets
- The remainder is saved as part of the bit count for the next frame's transmission.
- If the next frame is not ready for transmission the Deference process ignores the remainder, and initializes the ifsStretchCount variable to zero

BitTransmitter Process

```
while transmitting do
    begin
        if (currentTransmitBit > lastTransmitBit) then TransmitBit(extensionBit)
        else
            if extendError then
                TransmitBit(extensionErrorBit) {jam in extension}
            else
                begin
                    TransmitBit(outgoingFrame[currentTransmitBit]);
                    if ifsStretchMode then
                        begin
                            ifsStretchCount := ifsStretchCount + 1;
                            if (ifsStretchCount = ifsStretchRatio) then
                                begin
                                    ifsStretchSize := ifsStretchSize + 1;
                                    ifsStretchCount := 0
                                end
                        end
                end;
```

BitTransmitter Process - How it works

- BitTransmitter counts the # of bits transmitted in the current frame, by incrementing ifsStretchCount.
- This variable is initialized by the Deference process to either a value of zero or to a value in the range between zero and (ifsStretchRatio - 1)
- This depends on the variable's value at the completion of transmission of the previous frame and the time the current frame's transmission has been initiated.
- When this variable reaches the value of ifsStretchRatio, the ifsStretchSize variable is incremented, which indicates to the Deference process that the minimum interframe spacing should be extended by one more octet
- This same procedure occurs during PhysicalSignalEncap

Additional Changes to Pascal Code

- Pascal code has been completely updated
  - All figures and references are defined to provide speed independence
  - Definitions all use bit times
  - The code is now 802.1d compliant
- FCS Passing feature is now fully specified in the Pascal Code

MAC - FCS Passing

- In the older versions of the 802.3 standard the MAC specifies that an implementer is always required to:
  - Insert the source address for a transmit frame.
  - Generate the CRC for a transmit frame.
  - Strip the CRC for a receive frame.
- There are several problems with these requirements:
  - There are no known implementations that do number one
  - A lot of implementations (except some NICs) don't do 2 & 3.
  - The 802.1D standard for transparent bridges, violated these requirements,
    - The reason is that these bridges must forward the frames unmodified.
    - This allows for end-to-end CRC coverage of the headers and data payloads.
  - The result was an inconsistency between two major IEEE standards
- The 802.3ad Link Aggregation Standard forced a change
  - LACP protocol initiates frames at the Link Aggregation sublayer (above the MAC)
  - It has a source address that cannot be inserted or substituted by one of the underlying MACs

MAC - FCS Passing Feature
Pascal Variables

- passReceiveFCSMode: Boolean;
  - Enables passing of the frame check sequence field of all received frames from the MAC sublayer to the MAC client
  - passReceiveFCSMode is a static variable;
  - Does not change between invocations of the Initialize procedure
- fcsParamValue:
  - FCS passed from MAC client
  - When fcsParamPresent is true: fcsField := fcsParamValue
  - Otherwise fcsField := CRC32(outgoingFrame)
- fcsParamPresent
  - If the MAC client chooses to generate the FCS field for the frame, it passes this field to the MAC sublayer via the fcsParamValue parameter.
  - When true, TransmitDataEncap uses the fcsParamValue parameter as the frame check sequence field for the frame

FCS Passing - TransmitDataEncap

```
procedure TransmitDataEncap;
begin
    with outgoingFrame do
        begin {assemble frame}
            view := fields;
            destinationField := destinationParam;
            sourceField := sourceParam;
            lengthOrTypeField := lengthOrTypeParam;
            if fcsParamPresent then
                begin
                    dataField := dataParam; {No pad if FCS passed from MAC client}
                    fcsField := fcsParamValue {Use the FCS passed from MAC client}
                end
            else
                begin
                    dataField := ComputePad (dataParam);
                    fcsField := CRC32(outgoingFrame)
                end;
```

FCS Passing - How It Works

- Frame Assembly - TransmitDataEncap - fcsParamPresent:=True
  - MAC sublayer uses the client-supplied FCS value, if present.
  - If the MAC client provides FCS, the padding field shall also be provided by the MAC client, if necessary
- Frame Assembly - TransmitDataEncap - fcsParamPresent:=False
  - The sublayer uses the dataParam to compute the pad if necessary
  - The sublayer independently computes the frame check sequence value.
- Frame Reception - RecieveDataDecap - fcsParamPresent:=True
  - The fcsParamValue is set by the variable fcsField
  - The fcsParamPresent variable is set by the variable passReceiveFCSMode
  - MAC sublayer passes the FCS of all received frames to the MAC client
  - MAC sublayer leaves the padding and data field of the frame intact.

Conclusion
10 Gigabit Ethernet will succeed

- Compatible - from 1 Mb to 10 Gb
- Broader application - from Lan to Wan
- Scalable using link aggregation
- Cost effective