SDC
STORAGE DEVELOPER CONFERENCE
SNIA SANTA CLARA, 2015

SMB 3.1.1

Greg Kramer
Principal Software Engineer
Microsoft

Dan Lovinger
Principal Software Engineer
Microsoft

Agenda

1. Dialect Changes
2. Extensible Negotiation
3. Preauthentication Integrity
4. Cluster Dialect Fencing
5. Cluster Client Failover (CCF) v2
6. Encryption Improvements
7. Future Directions
8. Questions

1 - Dialect Changes

- Dialects now written “Major.Minor.Rev”

  - Simplify to “Major.Minor” if revision is 0
  - Examples: 2.0.2, 2.1, 3.0, 3.0.2, 3.1.1, …, 255.15.15 (Windows Server 3015 - Quantum Computing Edition) ☺
  - Already updated in protocol document and UI

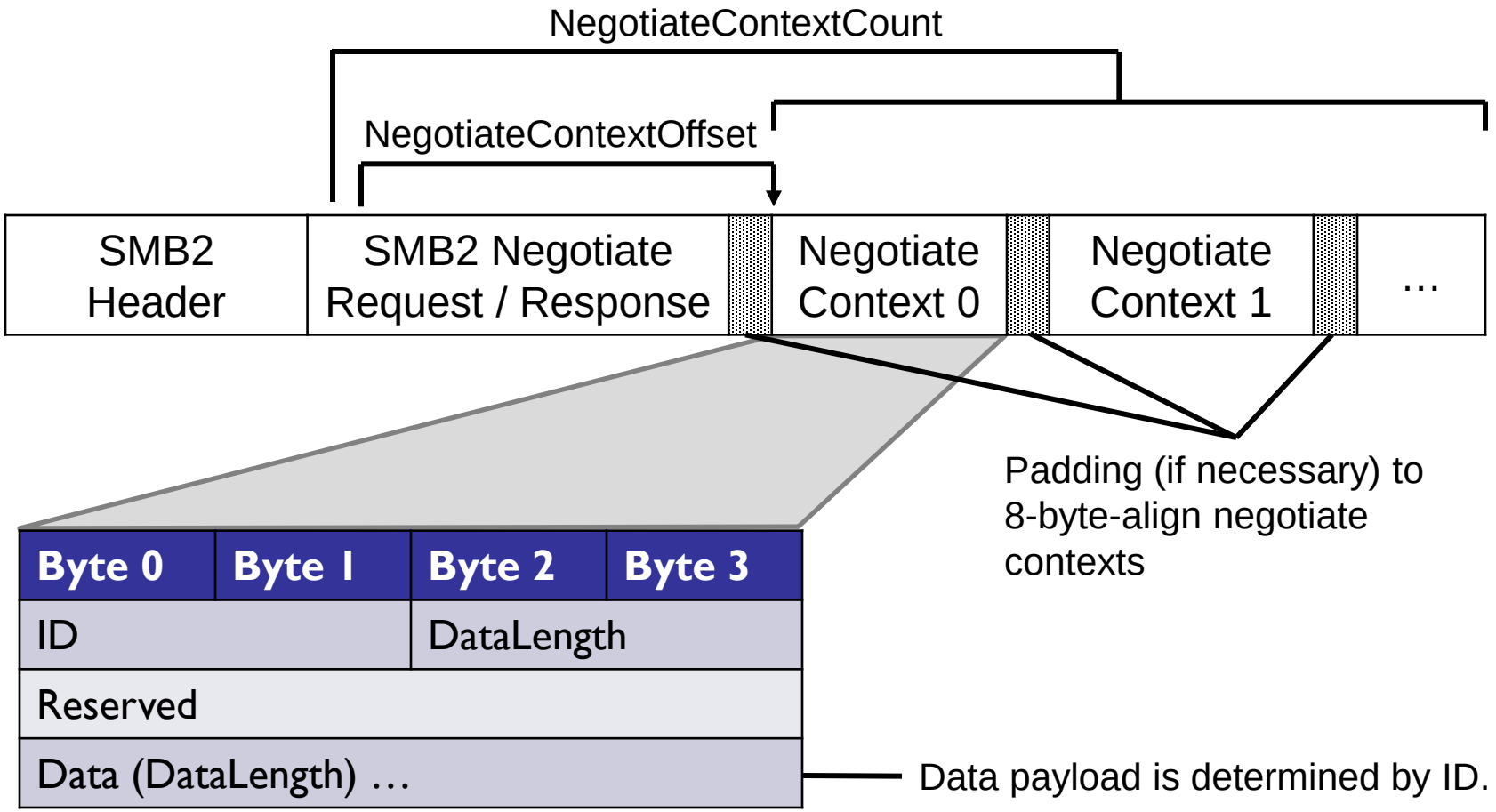
1 - Dialect Changes…

- The Windows 10 SMB dialect is 3.1.1 (0x0311)
  - At SDC 2014 (Windows 10 Preview) it was 3.1 (0x0310)
  - Very minor changes compared to SMB 3.1
  - Differences from 3.1 dialect are *called out in italic red text*
  - SMB 3.1 is now unsupported and will be rejected
  - We expect, but cannot promise, that the Windows Server 2016 SMB dialect will also be 3.1.1.

2 - Extensible Negotiation

- How to negotiate arbitrarily complex connection capabilities?
  - Few unused bits left in the negotiate request / response
- SMB 3.1.1 Extensible Negotiation
  - Exchange additional negotiate information via negotiate contexts (same idea as the existing create contexts).
  - Repurpose unused fields in negotiate request / response as *NegotiateContextOffset* and *NegotiateContextCount* fields.
  - Add list of negotiate contexts to end of existing negotiate request / response messages.

2 - Negotiate Contexts

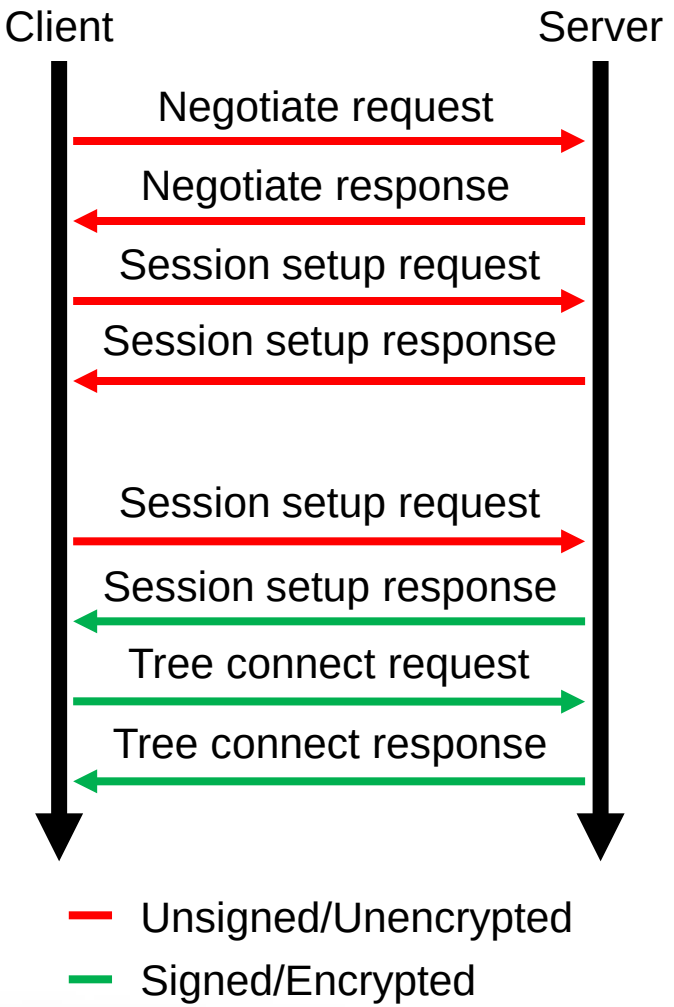


2 - Key Points

- Client sends negotiate contexts only if it supports the 3.1.1 dialect.
- Server sends negotiate contexts only if it selects 3.1.1 as the connection's dialect.
- Receiver must ignore unknown negotiate contexts.
  - Allows new contexts to be added without requiring a new dialect.
- SMB 2/3 server implementations must be willing to accept negotiate requests that are larger than the SMB2_HEADER + SMB2_REQ_NEGOTIATE + Dialects array.
  - A client does not know apriori whether a server supports SMB 3.1.1, so must assume that it does and send negotiate contexts.
  - Windows accepts negotiate requests as large as 128 KiB

3 – Preauthentication Integrity

- How to protect negotiate / session setup messages from tampering?
  - No protection prior to SMB 3.0
  - SMB 3.0.x Negotiate Validation doesn't protect negotiate contexts or session setup messages.
- SMB 3.1.1 Preauthentication Integrity
  - Provides end-to-end, dialect agnostic protection.
  - Session's secret keys derived from hash of the preauthentication messages.
  - Server signs final session setup response.
  - *Client signs or encrypts tree connect requests*.
  - Signature validation/decryption of authenticated messages will fail in case of preauthentication message tampering.



3 - Selecting the Hash Function

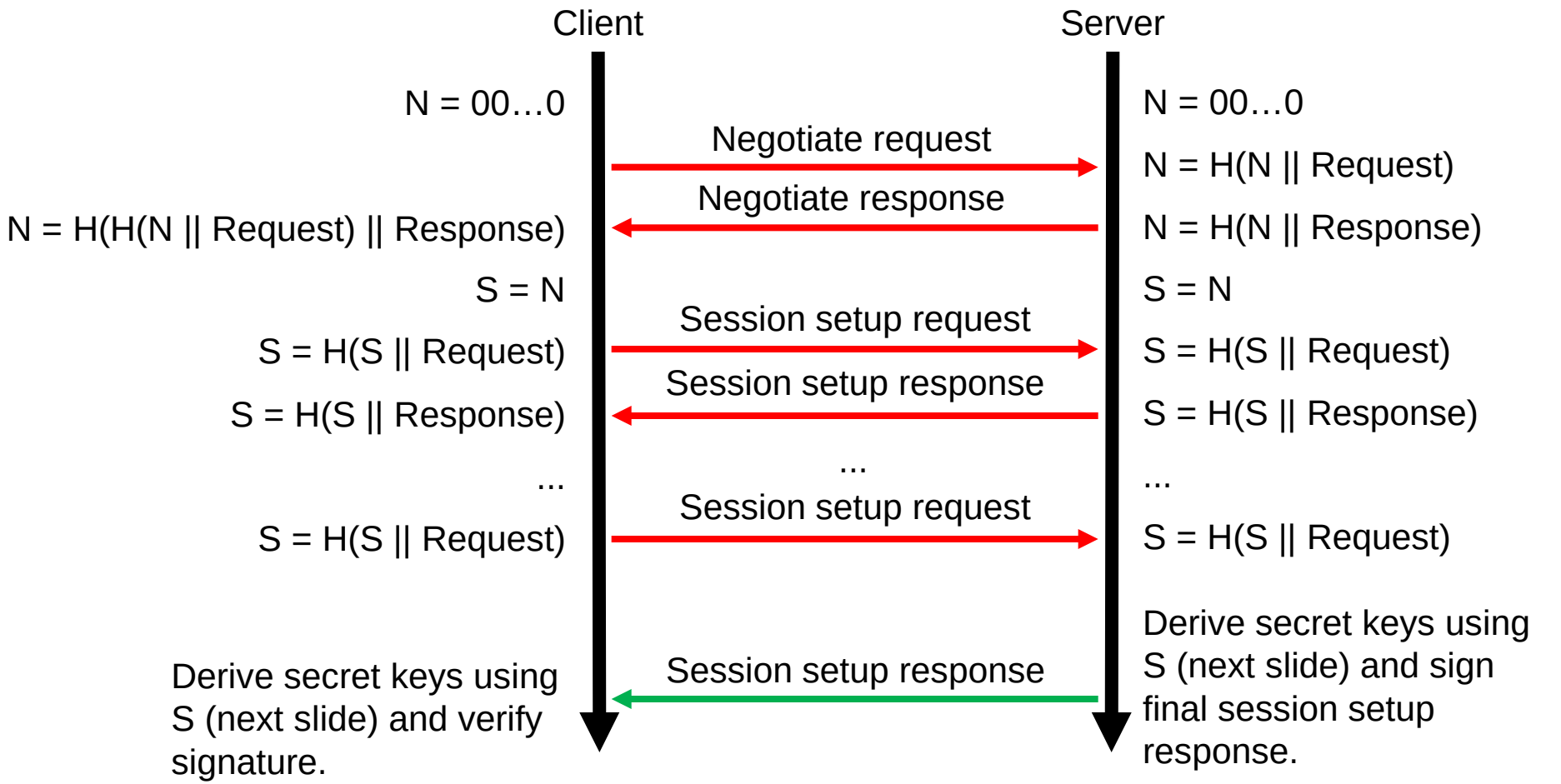
- SMB 3.1.1 client and server exchange mandatory negotiate contexts for each connection.
- Client's negotiate context specifies a set of supported hash functions.
- Server's negotiate context specifies the selected hash function.
- SHA-512 is currently the only supported hash function.

SMB2_PREAUTH_INTEGRITY_CAPABILITIES
(Negotiate Context ID: 0x0001)

Byte 0	Byte 1	Byte 2	Byte 3
HashAlgorithmCount		SaltLength	
HashAlgorithms			
…			
Salt			
…			

- Preimage attack resistance is provided by a salt value that the client and server generate via a secure PRNG per request/response.

3 - Computing the Integrity Hash Value



H(x): hash value of bit string x using the negotiated hash function
A || B: concatenation of bit strings A and B

3 - Deriving Secret Keys in SMB 3.1.1

DerivedKey = KDF[1](SessionKey, Label[2], Context)

Derived Key	Label
Application Key	“SMBAppKey”
Signing Key	“SMBSigningKey”
Client to server cipher key	“SMBC2SCipherKey”
Server to client cipher key	“SMBS2CCipherKey”

Session’s final preauthentication integrity hash value (S)

1. KDF is SP108-800-CTR-HMAC-SHA256 (same as SMB 3.0.x)
2. Note that KDF labels have changed since SMB 3.0.x
3. Key derivation for pre-3.1.1 dialects unchanged

3 - Key Points

- Preauthentication Integrity is mandatory for SMB 3.1.1.
- Preauthentication Integrity supersedes SMB 3.0.x Negotiate Validation for SMB 3.1.1 connections.
- Session setup hashes are only calculated for master and binding session setup exchanges, not reauthentication.
  - Reauthentication does not result in new keys

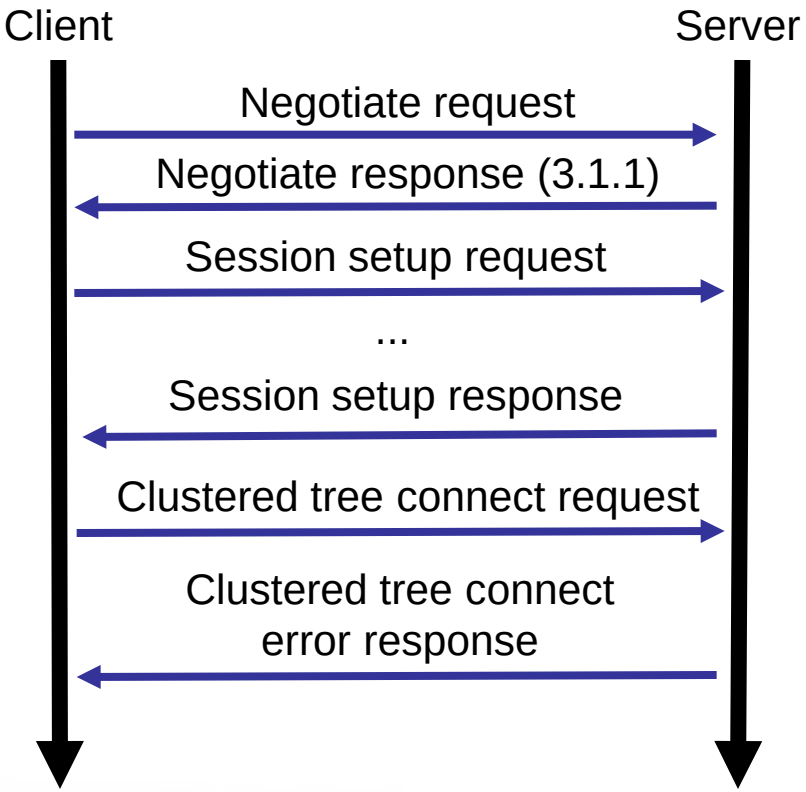
4 – Cluster Dialect Fencing

- How to support Cluster Rolling Upgrades?
  - Cluster nodes with upgraded OS may support a higher SMB dialect than nodes that have not yet been upgraded (3.1.1 vs. 3.0.2).
  - Transparent failover requires all cluster nodes to support the same set of dialects so that handles opened on one node can fail over to any other node in the cluster.
- SMB 3.1.1 Cluster Dialect Fencing
  - Define a maximum SMB cluster dialect that all nodes support.
  - Fence access to cluster shares based on the maximum SMB cluster dialect.
  - Fenced clients instructed to reconnect at a cluster-supported dialect.

4 - Fencing Clustered Tree Connects

An SMB 3.1.1 client accesses a clustered file share on an SMB 3.1.1 server that is a member of a cluster whose maximum SMB cluster dialect is 3.0.2.

1. Client negotiates 3.1.1, authenticates then issues tree connect.
2. Server fails tree connect request with an extended error (status = 0xC05D0001) whose data payload indicates the maximum cluster-supported dialect (3.0.2).
3. Client disconnects, reconnects with new Client GUID, negotiates 3.0.2, authenticates, then reissues tree connect.



4 – Tree Connect Request Changes

Byte 0	Byte 1	Byte 2	Byte 3
Structure Size		~~Reserved~~ Flags	
PathOffset		PathLength	
Buffer			
…			

Reserved field renamed to Flags:

Value	Meaning
0x0001	Client has already successfully connected to a clustered file share on this server at the current SMB dialect.

- Once a client has successfully connected to a clustered share it must set the CLUSTER_RECONNECT (0x0001) flag on all subsequent clustered tree connect requests to the same server.
  - Addresses a race condition when the maximum SMB cluster dialect has been raised but some nodes have not yet begun allowing the new, higher dialect.

4 - Key Points

- Dialect fencing only affects clustered share access.
  - Clients can still access non-clustered shares using dialect X even if the maximum SMB cluster dialect is < X.
  - Can’t mix clustered and non-clustered access on same connection.
- Client implementation should protect against infinite loop of tree connect failure, disconnect, reconnect, tree connect failure, …

5 - Cluster Client Failover (CCF) v2

- CCF v1 Overview
  - Introduced with SMB 3.0 for clustered applications using SMB 3.0 storage
  - Permits clustered application to tag an open with *ApplicationInstance* identifier
  - An open issued by a different client with the same *ApplicationInstance* indicates workload has transitioned to a new node, so old opens are closed allowing new node to reopen handles.

5- Cluster Client Failover (CCF) v2…

- How to handle an application cluster partition?
  - Cluster loses network access to a node running an application but that node can still access storage.
  - Cluster restarts application on a new node.
  - Application now running on two nodes, fighting over access to handles.

- SMB 3.1.1 CCF v2
  - The cluster knows which node should be hosting an application. Along with the ApplicationInstance, it provides an ApplicationInstanceVersion to convey this knowledge to the application node.
  - The ApplicationInstanceVersion is increased every time the application is moved to a new node.

5 - Cluster Client Failover (CCF) v2…

- SMB 3.1.1 Client must
  - Pass ApplicationInstanceVersion alongside ApplicationInstance on create
  - It should attempt to keep the handle alive until it receives a non-ambiguous status code from the server indicating it has been superseded by another node (or the handle has timed out)
- SMB 3.1.1 Server must
  - Compare the ApplicationInstanceVersion on an invalidating open.
    - If the version is higher, the existing open should be orphaned as normal.
    - If the version is lower, the incoming open is failed with a non-ambiguous status code indicating it has been superseded.

5 - Cluster Client Failover (CCF) v2…

- To interact with older (pre-SMB 3.1.1) clients
  - Opens without a version are assumed to be version 0
  - A version 0 open will successfully invalidate other version 0 opens
  - Otherwise, the same rules apply

6 - Encryption Improvements

- SMB 3.0.x mandates the AES-128-CCM cipher
  - What if a different cipher is required for performance, regulatory requirements, etc?
  - What if a cipher is compromised and needs to be retired?
- SMB 3.1.1 Encryption Improvements
  - Ciphers are negotiated per-connection
  - Added support for AES-128-GCM
  - ~~*Clients can mandate that sessions be encrypted even if the server does not require encryption.*~~
    - *No protocol changes necessary. Client just requires signing during negotiation / session setup then issues only encrypted requests.*

6 – Negotiating a Cipher

- SMB 3.1.1 client and server exchange negotiate contexts for each connection if they support encryption.
- Client's negotiate context specifies a set of supported ciphers in order from most to least preferred.

SMB2_ENCRYPTION_CAPABILITIES
(Negotiate Context ID: 0x0002)

Byte 0	Byte 1	Byte 2	Byte 3
CipherCount		Ciphers	
...			

- Server's negotiate context specifies the selected cipher.
  - Selection policy is server's choice: client-preferred, server-preferred, etc.
  - Reserved cipher ID 0x0000 indicates that the client and server have no common cipher.
  - No SMB2_ENCRYPTION_CAPABILITIES context in server response indicates that the server does not support encryption.
- Encryption capabilities flag is never set in an SMB 3.1.1 Negotiate Response.

6 - Transform Header Changes

Byte 0	Byte 1	Byte 2	Byte 3
ProtocolId			
Signature			
…			
…			
…			
Nonce			
…			
…			
…			
OriginalMessageSize			
Reserved		Flags	
SessionId			
…			

Nonce size determined by cipher:

Cipher	Nonce Size (bytes)
AES-128-CCM	11
AES-128-GCM	12

EncryptionAlgorithm field renamed to Flags:

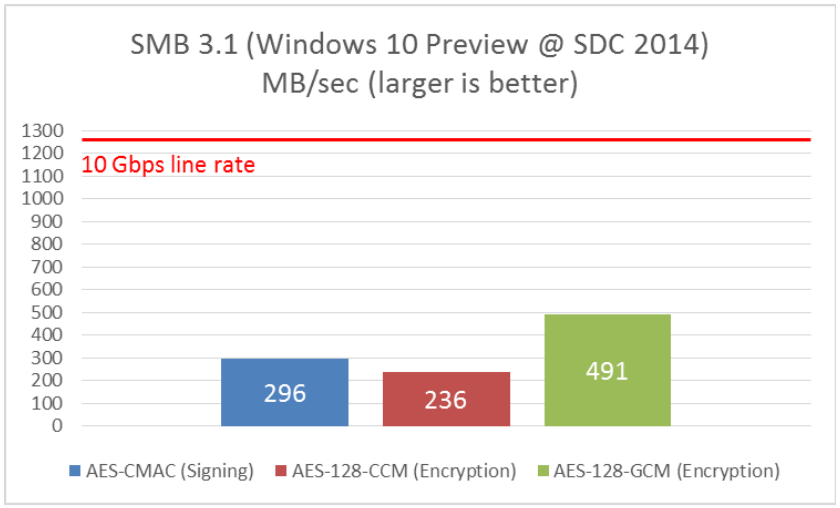
Value	Meaning
0x0001	Payload is encrypted using cipher negotiated for the connection

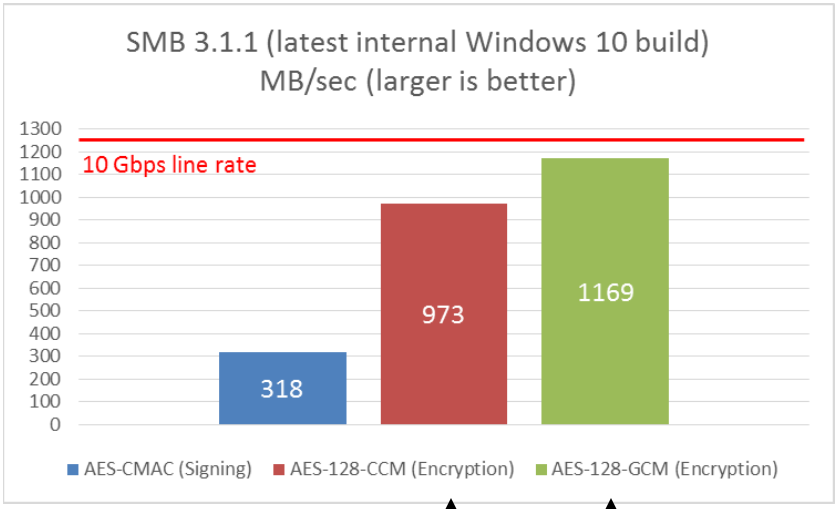
6 - Performance

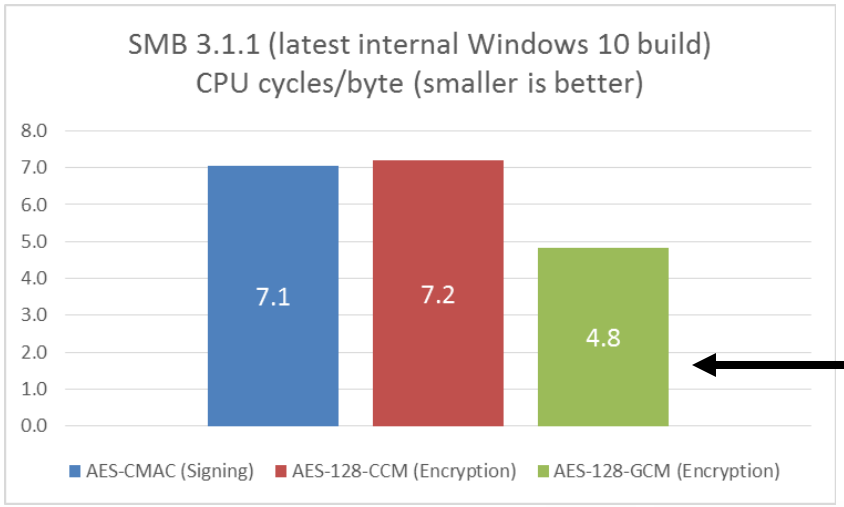
- Examine large file copy performance
- SMB can copy at 10 Gbps line rate when not using signing or encryption.

Test configuration (client and server)	
CPU	2x Intel Xeon E5-2660 @ 2.2 GHz (16 physical cores, HyperThreading disabled)
OS Power Profile	High Performance
Network Adapter	1x Intel Ethernet Server Adapter X520 @ 10 Gbps
Storage Device	NVMe
Storage Workload	File copy (1 thread doing 8 async 1 MiB writes)

6 – Performance…







4.12x faster than SMB 3.1

2.38x faster than SMB 3.1

No protocol changes required!

33% fewer cycles/byte than AES-128-CCM

These improvements will ship in the next Windows Server 2016 preview and the next Windows 10 client release.

6 - Key Points

- AES-CCM required for SMB 3.0.x compatibility.
- AES-GCM provides **significant** performance / efficiency improvements and should be supported.
- Session binding (multichannel) requires all of a session's channels to negotiate the same cipher as the session's original connection.

7 – Future Directions

Some of the following slides discuss experimental, protocol changes. Microsoft makes no promise that these changes will ship.

7 – Improving SMB Signing Performance

- SMB Encryption using AES-GCM is much faster / more efficient than SMB Signing.
  - But what if we only need integrity?
  - Why spend CPU cycles encrypting data if we don't need privacy?
- Can SMB Signing be made faster / more efficient than SMB Encryption?

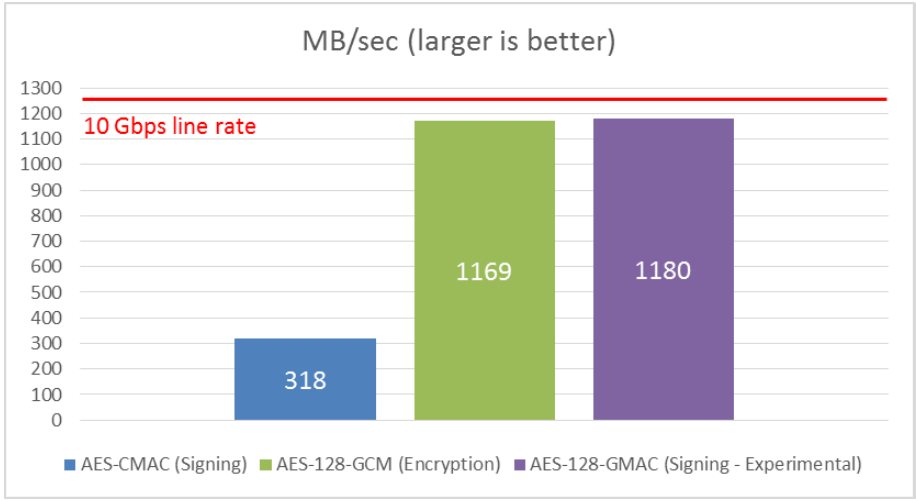
7 – AES-GMAC Signing

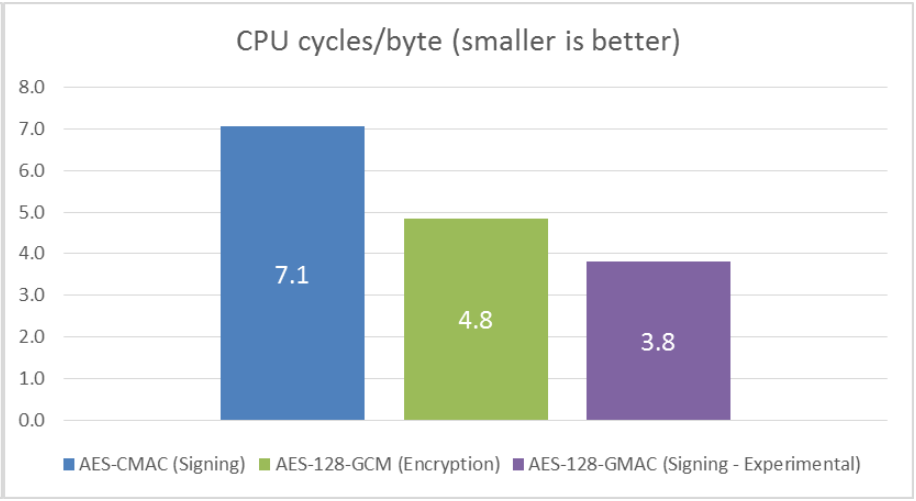
- AES-GMC
  - Authenticated encryption (integrity + privacy)
  - Fast / efficient
  - Used in SMB 3.1.1 for SMB Encryption
- AES-GMAC
  - Integrity-only mode of AES-GCM encryption
  - Should be faster / more efficient than AES-GCM since it does less work.
- Meet Aaron Friedlander
  - Microsoft 2015 summer intern from Carnegie Mellon
  - Prototyped AES-GMAC signing support for SMB 3.1.1
  - Did a really great job on a complex code base with no prior kernel development experience.

7 – Supporting AES-GMAC in SMB 3.1.1

- Define a new signing capabilities negotiate context that an SMB 3.1.1 client and server use to negotiate a signing algorithm on a per-connection basis.
  - Prototype SMB 3.1.1 clients and servers interoperate with standard SMB 3.1.1 clients and servers.
  - Proof that negotiate contexts allow features to be added without requiring a new dialect.
- Refactor the encryption code paths to handle both authenticated encryption (AES-CCM/GCM) as well as AES-GCM in signing-only mode (AES-GMAC)
- Add a new transform header flag value to indicate that the payload is signed, not encrypted.

7 – AES-GMAC file copy performance





- AES-GMAC results in significant performance improvements!
  - 46% reduction in Cycles/Byte compared to AES-CMAC
  - 21% reduction in Cycles/Byte compared to AES-GCM

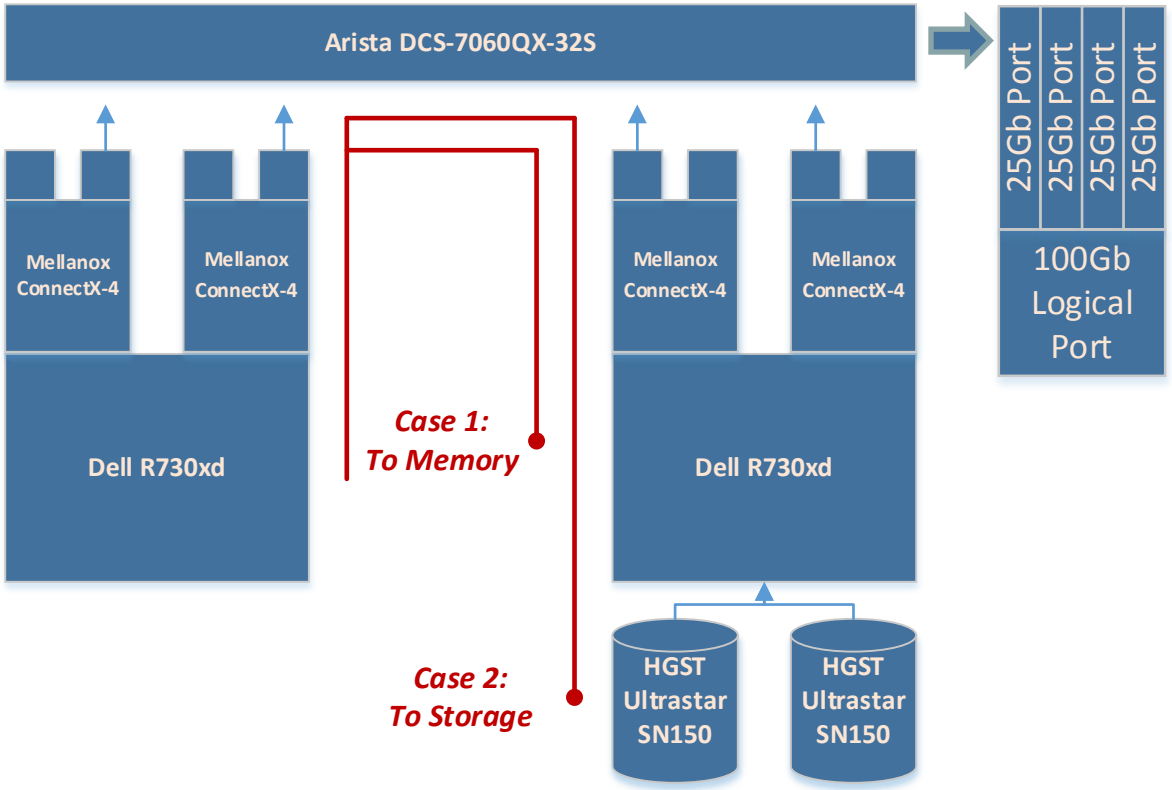
- Prototype focused on functional correctness not performance
  - We identified several fairly easy improvements that could be made to further decrease CPU cycles/byte.

7 – Faster

100 x2

7 - Dual 100GbE Multi-Vendor Test Configuration

- Windows Server 2016 TP3
  - SMB 3.1.1 + SMB Direct
- Arista DCS-7060CX-32S 32port 100Gb Switch
- Dell R730xd hosts
  - 2x E5-2660v3 (2.6Ghz 10c20t)
  - 256GiB DDR4 2133MT/s (16x 16GB)
  - 2x HGST UltraStar SN150 NVME (1.6TB PCIe 3.0 x4)
  - 2x Mellanox ConnectX-4 1 Port Connected (PCIe 3.0 x16)
  - Mellanox Copper 100Gb Cable



7 - Case 1: SMB3 to Remote Memory Cache

Extreme Network Bandwidth

RDMA Activity	Mellanox ConnectX-4 VPI Adapter	Mellanox ConnectX-4 VPI Adapter #3
RDMA Accepted Connections	0.000	0.000
RDMA Active Connections	2.000	2.000
RDMA Completion Queue Errors	0.000	0.000
RDMA Connection Errors	0.000	0.000
RDMA Failed Connection Attempts	0.000	0.000
RDMA Inbound Bytes/sec	---	---
RDMA Inbound Frames/sec	11,010,314.645	10,933,119.858
RDMA Initiated Connections	34.000	34.000
RDMA Outbound Bytes/sec	---	80,032,685.356
RDMA Outbound Frames/sec	902,128.513	894,007.535

SMB Client Shares	\\431217c10-21\c$
Avg. Bytes/Read	524,288.000
Avg. Bytes/Write	0.000
Avg. Data Bytes/Request	524,288.000
Avg. Data Queue Length	8.462
Avg. Read Queue Length	8.462
Avg. sec/Data Request	0.000
Avg. sec/Read	0.000
Avg. sec/Write	0.000
Avg. Write Queue Length	0.000
Credit Stalls/sec	0.000
Current Data Queue Length	7.000
Data Bytes/sec	22,382,317,301.6365
Data Requests/sec	42,690.882
Metadata Requests/sec	0.000
Read Bytes/sec	22,382,841,588.1910
Read Requests/sec	42,691.882
Write Bytes/sec	0.000
Write Requests/sec	0.000

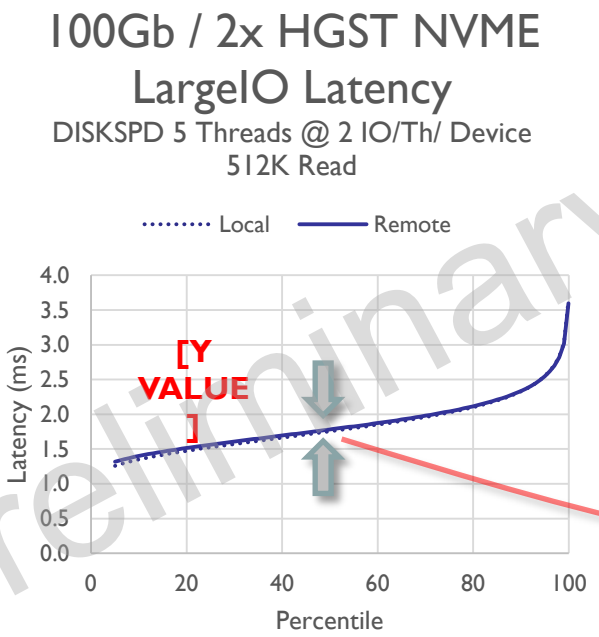
Theoretical ~11.5GB/s/link, ~23GB/s total

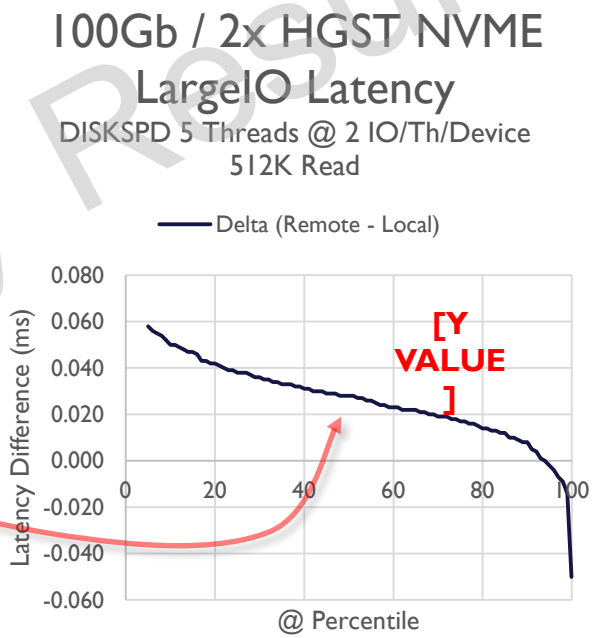
22.3GB/s!

7 - Case 2: SMB3 to HGST NVME Storage

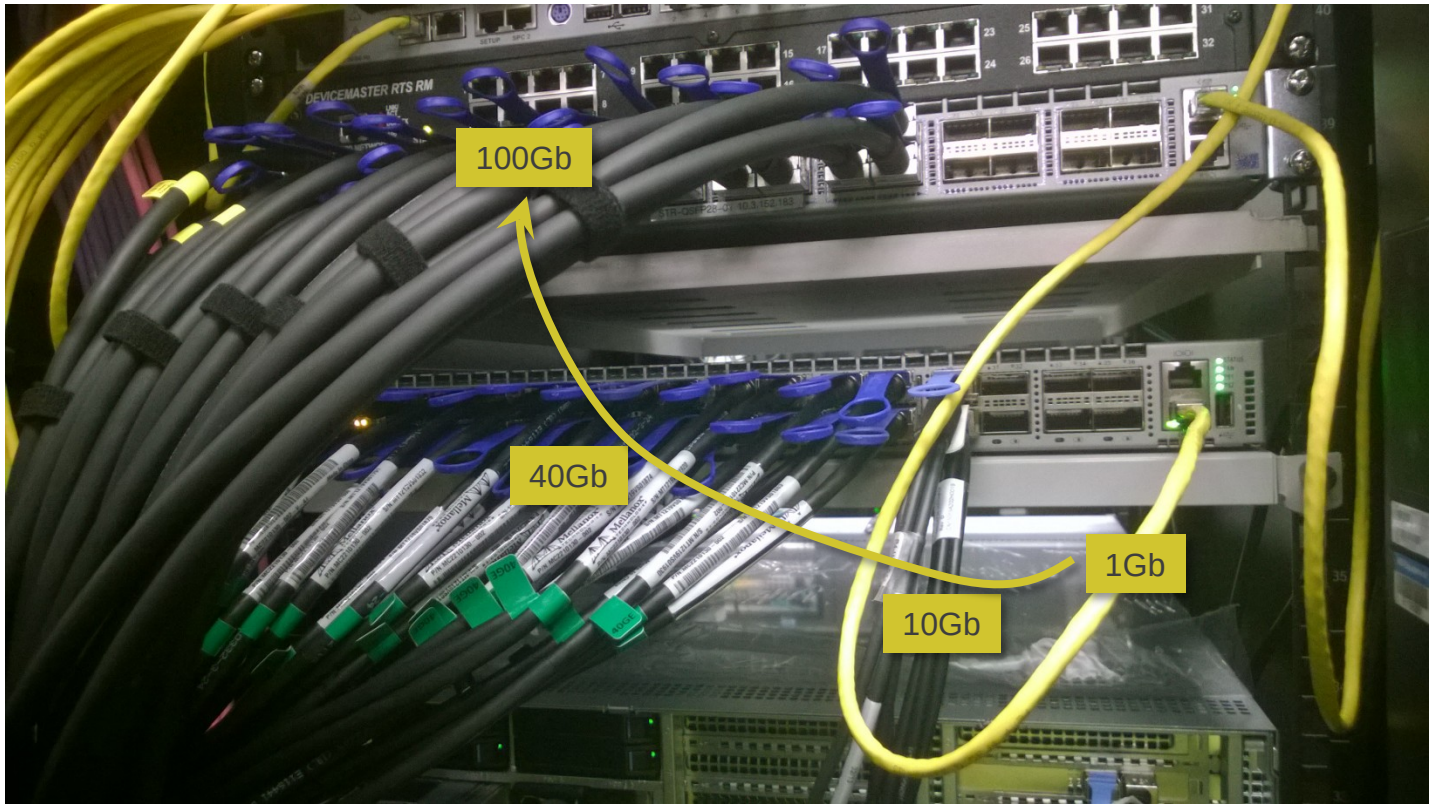
Minimal Wire Latency to Remote Storage

- Load to dual HGST NVME on Remote System
- Latency measured ***End-to-End over SMB3*** to DISKSPD
- Near-saturation NVME devices @ ***5.7GB/s***
- Only ***28us latency introduced on the wire for the median IO***





7 - 16 Years of Ethernet In The Rack



Questions?