**High Performance Network Adapters and Drivers in Windows**

**Contents:**

[Introduction](http://www.htsns.com/LoginTest/msnet/msdoc/high%20performance%20network%20adapters%20and%20drivers%20in%20windows.html#Introduction)

[Network Adapter Hardware](http://www.htsns.com/LoginTest/msnet/msdoc/high%20performance%20network%20adapters%20and%20drivers%20in%20windows.html#Hardware)

[Network Adapter Driver](http://www.htsns.com/LoginTest/msnet/msdoc/high%20performance%20network%20adapters%20and%20drivers%20in%20windows.html#Driver)

[Scatter-Gather DMA](http://www.htsns.com/LoginTest/msnet/msdoc/high%20performance%20network%20adapters%20and%20drivers%20in%20windows.html#Scatter-Gather)

[Auto-tuning](http://www.htsns.com/LoginTest/msnet/msdoc/high%20performance%20network%20adapters%20and%20drivers%20in%20windows.html#Auto)

[Call to Action](http://www.htsns.com/LoginTest/msnet/msdoc/high%20performance%20network%20adapters%20and%20drivers%20in%20windows.html#Call)

네트워킹 성능은 전반적인 시스템 성능에서 가장 중요한 요소의 하나로 점점 되어가고 있다. 네트워킹 성능에 영향을 주는 많은 요소들은 다음의 3가지 종류가 있다.: 네트워크 어댑터 하드웨어와 드라이버 성능, 네트워크 스택 성능, 응용프로그램이 네트워크 스택과 상호작용하는 방법. 이 기사는 Microsoft® Windows® 2000, Windows XP, Windows .NET Server 에서 네트워크 어댑터 하드웨어와 드라이버 성능에 관계된 이슈들을 설명한다.

**Introduction   
[Back to Top](http://www.htsns.com/LoginTest/msnet/msdoc/high%20performance%20network%20adapters%20and%20drivers%20in%20windows.html#top)**

네트워킹 성능은 처리량과 응답 시간으로 측정된다. 시스템 자원들을 넘게 이용하지 않고 최적의 성능 연산 포인트를 이루는 것은 중요하다. 그런 최적의 연산 포인트를 이루기 위해서, 네트워크 어댑터 벤더들은 몇몇 분야들을 조사할 필요가 있다.

* · 어느 네트워크 어댑터 하드웨어 능력들이 수행되는가
* · 임계 경로 길이와 임계 경로 길이와 scalability에 대한 네트워크 어댑터 드라이버 수행
* · 자동-터닝 하는 것을 허용하는 동적인 적당한 하드웨어와 소프트웨어 인자들

**Network Adapter Hardware   
[Back to Top](http://www.htsns.com/LoginTest/msnet/msdoc/high%20performance%20network%20adapters%20and%20drivers%20in%20windows.html#top)**

네트워크 어댑터에서 수행하기위해서 어느 하드웨어 함수들을 결정할지 항상 교환이 있다. 인터럽트 완화를 허용하는 task offload features을 추가하는 것, 동적으로 하드웨어를 tuning on하는 것, PCI 버스의 사용을 개선하는 것, 점보 프레임들을 지원하는 것을 고려하는 것은 점점 중요해지고 있다. 최상의 성능을 요구하는 구성에서 사용될 high-end 네트워크 어댑터에 대해서 특히 중요성이 있다.

**Task Offload Features**

Task offload features은 드라이버나 펌웨어에서 수행될 수 있다. 최적의 성능에 대해서, 하드웨어에서 Task offload 를 수행하는 것은 최상이다. 윈도우는 다음의 세 개의 Task offload features을 인에이블한다. Task offload에 대한 문서는 윈도우 SDK에서 찾을 수 있다.

* TCP and IP Checksum Offload: 가장 일반적인 네트워크 트래픽에 대해서, 네트워크 어댑터 하드웨어에서 체크섬 계산을 오프로딩하는 것은 바이트 당 요구되는 CPU사이클의 수를 감소시키므로 상당한 성능 이점을 제공한다. 체크섬 계산은 두 가지 이유로 네트워킹 스택에서 가장 고가의 기능이다.:
  + · 긴 패스 길이에 도움이 된다.
  + · 캐쉬가 churning effects하게 한다.(일반적으로 보내는 쪽에서)

보내는 자에서 체크섬 계산을 오프로딩하는 것은 호스트 CPU의 부하를 감소시키고 캐쉬 효과를 증가시키므로 전반적인 시스템 성능을 개선한다.

윈도우 성능 Lab에서, network-intensive가 workloads하는 동안 체크섬 오프로드될 때, TCP 처리량이 19% 개선되게 측정되었다. 이 개선의 분석은 전체 개선의 11%가 the path length reduction로 발생했고 8%는 cache's effectiveness의 증가로 발생했다.

수신자에서 체크섬을 오프로딩하는 것은 보내는 자에서 체크섬을 오프로딩하는 것과 같은 이점을 가진다. 증가된 이점은 소켓 프락시 서버와 같은 클라이언트와 서버 둘 다에서 행하는 시스템에 보여질 수 있다. 클라이언트 시스템 같은 CPU가 바쁠 필요가 없는 시스템에서, 체크섬 오프로딩의 이점은 개선된 처리량보다는 더 좋은 네트워크 응답 시간으로 보여진다.

* 윈도우는 TCP-up 에서 64K에서 MTU보다 더 큰 Maximum Segment Size (MSS)을 알리기 위해서 네트워크 어댑터/드라이버에 대한 능력을 제공한다. 이것은 TCP가 드라이버에서 64K까지 버퍼를 할당하는 것을 허용한다. 그리고 그 드라이버는 네트워크 MTU에 안에서 적당한 패킷들로 큰 버퍼를 나눈다.

TCP 세그먼테이팅 작업은 호스트 CPU 대신에 네트워크 어댑터/드라이버 하드웨어에 의해서 행해진다. 이것은 네트워크 어댑터 CPU가 추가적인 작업을 다룰 수 있다면 상당한 성능 개선의 결과가 발생한다.

테스트된 많은 네트워크 어댑터들에 대해서, 호스트 CPU가 네트워크 어댑터 하드웨어보다 더 파워풀 했을 때 순전히 네트워킹 활성화에 대해서 보여지는 개선은 아주 작다. 그러나, 일반적인 business workloads에 대해서, 처리량의 9%까지 전반적인 시스템 성능 개선이 측정된다. 이유는 호스트 CPU가 처리를 실행하기위해서 대부분의 사이클을 사용하기 때문이다. 이 경우에, 하드웨어에서 TCP 세스먼테이션을 오프로딩하는 것은 세그먼테이션의 부하로부터 호스트 CPU의 부하는 자유롭게 한다. 그것의 여분의 사이클은 더 많은 처리량을 수행하는 것을 허용한다.

* **IP Security (IPSec) Offload:** 윈도우는 네트워크 어댑터 하드웨어에서 IPSec의 암호화 작업을 오프로드하는 능력을 제공한다. 특히 3 DES는 매우 높은 cycles/byte ratio를 가진다. 그래서, 네트워크 어댑터 하드웨어에서 IPSec을 오프로딩하는 것이 보안 인터넷과 VPN 테스트에서 30%의 성능 향상이 측정된 것은 놀랄 일이 아니다.

**Interrupt Moderation**

간단한 네트워크 어댑터는 패킷의 도착에서나 패킷 보내기 요구의 시그널 완료에서 호스트에 하드웨어 인터럽트를 생성한다. 인터럽트 대기시간이나 cache churning effects의 결과는 전반적인 네트워킹 성능에 오버헤드를 준다. 많은 시나리오에서(예를 들면, 많은 시스템 사용이나 네트워크 트래픽), 각각의 인터럽트에 대해서 여러 개의 패킷들을 처리하므로서 하드웨어 인터럽트의 손실을 감소시키는 것이 최상이다.

많은 네트워크 작업부하에서, 처리량에서 9%까지 성능 향상이 네트워크-집중적인(intensive) 작업부하에서 측정되었다. 그러나, 처리량 개선에 대해서만 인터럽트 Moderation 인자들을 조정하는 것은 응답 시간에서 성능 타격의 결과를 초래한다. 최적의 셋팅을 유지하기 위해서와 다른 작업부하에서 적응시키는 위해서, 뒤에 있는 기사인 자동-조정에서 설명되는 동적으로 적용된 인자들을 허용하는 것이 최상이다.

**PCI Bus Usage**

네트워크 어댑터 하드웨어 성능에서 가장 중요한 요소들 중에 하나는 PCI 버스의 효과적인 사용이다. 네트워크 어댑터의 DMA 성능은 같은 PCI 버스에 있는 모든 PCI 카드의 성능에 영향을 준다. 다음의 가이드라인은 PCI 사용을 최적화할 때 고려되어져야 한다.:

* · 사용하는 목적 패이지들을 한데 모음으로서 DMA 전송을 간소화한다.
* · 큰 덩어리(적어도 256바이트)에서 DMA 전송을 수행하므로서 PCI 프로토콜 오버헤드를 감소시킨다. 가능하다면, 전체 패킷들이 하나의 PCI 처리에서 전송되는 데이터의 흐름을 정해라. 그러나 전송이 어떻게 발생해야 하는지를 고려해라.; 예들 들면, 전송이 시작하기 전에 도착한 모든 데이터는 기다려선 안된다. 이것은 대기 사간을 증가시킬 것이고 추가적인 버퍼 공간을 소비한다.
* · 패킷의 마지막 적은 바이트들을 전송할 때 "clean up"하기위해서 적은 여분의 전송을 요구하는 것 보다는 DMA 패킷 전송에 추가적인 바이트들을 채우는게 더 좋다.
* · PCI 스펙에 의해서 고려되어지는 것같이 Memory Read, Memory Read Line, 과 Memory Read Multiple transactions을 사용한다.
* · 네트워크 어댑터 버스 인터페이스 하드웨어는 호스트 메모리 컨트롤러에서 제한들을 발견해야 하고 적당한 행위를 해야 한다. 예를 들면, 네트워크 어댑터 버스 인터페이스 하드웨어는 DMA 메모리 읽기에서 메모리-컨트롤러 pre-fetch limitations을 발견해야 한다. 하드웨어는 네트워크 어댑터의 일부분에서 과도한 반복을 발견해야 하고 처음에 호스트에 의해서 cut off 될 때 미래에 처리를 다시 하기 전에 시간을 증가해야 한다. 데이터의 다음 셋을 가져오기가 바쁠 때, 메모리 컨트롤러에서 처리들을 제안하는 것을 계속하는 것에서 포인트가 없다.
* · 기다리는 상태의 삽입을 최소화해라, 특히 데이터 전송동안에, 버스를 포기하는 것이 더 좋고 만약 하나 이상의 기다는 상태가 삽입되고 있다면 버스를 사용하는 또 다른 PCI 어댑터는 행해지는 어떤 작업을 가져야 한다.
* · 프로그램 I/O대신에 메모리 맵 I/O를 사용해라. 이것은 드라이버에 대해서 또한 TRUE이다.

**Jumbo Frame Support**

큰 Maximum Transmission Units (MTUs)과 더 큰 프레임 크기들, 특히 점보 프레임들을 지원하는 것은, 바이트 당 발생되는 네트워크 스택 오버헤드를 감소시킬 것이다. 20% TCP 처리량 증가는 MTU가 1514에서 9000으로 변화됐을 때 측정된다. 또한, 네트워크 스택으로부터 네트워크 드라이버로의 호출이 적게 되므로 CPU 이용률의 상당한 감소를 얻는다.

**Network Adapter Driver   
[Back to Top](http://www.htsns.com/LoginTest/msnet/msdoc/high%20performance%20network%20adapters%20and%20drivers%20in%20windows.html#top)**

네트워크 어댑터 드라이버의 성능 효과는 전반적인 네트워크 성능에서 중요한 요소이다. 네트워크 어댑터와 함께하는 성능 이슈들은 아래 3개 분야가 있다.: Path length, scalability, alignment.

**Path Length**

비록 드라이버로부터 드라이버까지 전송과 수신 경로들이 다르지만, 성능 최적화에 대해서 어떤 일반적인 규칙이 있다.

* · 일반적인 경로에 대해서 최적화해라. Kernprof.exe 툴은 필요한 정보를 푼 윈도우의 MSDN과 IDW 빌드로 제공된다. 개발자들은 최고로 CPU 사이클들을 소비하는 루틴들을 알아야 하고 호출되는 이 루틴들의 수나 이 루틴들에서 소비된 시간을 감소시키려고 시도해야 한다.
* · 네트워크 어댑터 드라이버가 과도한 시스템 자원들을 사용하지 않게 하기위해서 DPC에서 소비되는 시간을 감소시켜라. 이것은 전반적인 시스템 성능을 제공할 것이다.
* · 디버그 코드가 드라이버의 최종 릴리즈 버전에서 컴파일이 안되게 확실히 해라.; 이것은 과도한 코드를 실행하는 것을 피한다.

**Scalability Issues**

Scalability 이슈들은 주소를 지정하는데 매우 중요하다. 특히 SMP 시스템에 대해서. 가변적인 Scalability 이슈들은 이 장에서 논의된다.

**Partitioning**  
분할 작업은 프로세서들 사이에서 공유되는 데이터와 코드를 최소화하는데 필요하다. 분할 작업 도움은 시스템 버스 이용률을 감소시키고 프로세스 캐쉬의 효과를 개선한다. 공유하는 것을 최소화하기위해서 드라이버 개발자들은 다음을 고려해야 한다.:

* · 윈도우 DDK에서 설명된 것 같이 비직렬화된 미니포트로서 드라이버를 수행해라.
* · 전역과 공유된 데이터 접근을 감소시키기 위해서 프로세서당 데이터 구조체들을 사용해라. 이것은 동기화 없이 통계 카운터들을 유지하는 것을 허용한다. 이것은 코드 경로 길이를 감소시키고 성능을 증가시킨다. 필수적인 통계들에 대해서, 질의 시간에서 함께 추가되는 프로세서당 카운터들을 가진다. 만약 전역 카운터를 가져야 한다면, 카운터를 다루기 위해서 스핀락 대신에 interlocked연산들을 사용해라. 락 메키니즘들은(스핀락을 피해라) 뒤의 스핀락을 사용하는 것을 피하는 법에 대한 기사를 보아라.

이것을 편하게 하기위해서, KeGetCurrentProcessorNumber는 현재 프로세서를 결정하는데 사용될 수 있다. 프로세서당 데이터 구조체들을 할당할 때, 프로세서의 수를 결정하기위해서, KeQueryActiveProcessors함수는 사용된다. 이 API는 DDK의 미래 발표에서 포함에 대해 계획된다. 그러나 다음으로서 정의된다.:

KeQueryActiveProcessors는 시스템에서 활성화한 프로세서들을 설명하는 유사한 매스크를 리턴한다.

**KAFFINITY**

**KeQueryActiveProcessors (**

**);**

**Include**

*ntddk.h*

**Comments**  
유사한 매스크에서 전체 비트 셋은 시스템에서 활성화한 프로세서의 수를 지시한다. 드라이버들은 매스크안에 있는 모든 셋 비트들이 연속적일 것이라고 가정해선 안된다. 이유는 프로세서들은 운영 체제의 미래 발표에서 연속적인 수라는 보장이 없다. SMP 머신 안에서 프로세서들의 수는 0기반의 값이다.  
만약 캐쉬-라인 경쟁을 감소시키는 시도에서 프로세서당 데이터를 유지했다면, 윈도우 XP/윈도우 2000 드라이버는 KeQueryActiveProcessors를 호출할 수 있다.

KeQueryActiveProcessors은 어떤 IRQL에서도 호출될 수 있다.

See also: KeGetCurrentProcessorNumber

**False Sharing**  
False sharing는 프로세서들이 서로로부터 독립하는 공유된 변수를 요구한다. 그러나, 변수들이 같은 캐쉬라인에 있기 때문에, 프로세서들 사이에서 공유된다. 그런 상황에서, 캐쉬 라인은 어떤 변수들의 모든 접근에 대해서 프로세서들 사이에서 앞뒤로 움직일 것이다. 그것은 캐쉬 플러쉬와 리로드의 증가를 일으킨다. 이것은 시스템 버스 이용률을 증가시키고 전반적인 시스템 성능을 감소시킨다.

False sharing를 피하기 위해서, 중요한 데이터 구조체들은(스핀락, 버퍼 큐헤더, SList 등) NdisGetSharedDataAlignment를 사용하므로서 캐쉬 라인 바운더리들로 정렬시켜라..

**Locking Mechanisms (Avoiding Spinlocks)**  
스핀락의 사용은 적당히 사용되지 않는다면 성능에서 문제가 될 수 있다. 드라이버들은 적당한 인터락드 연산을 사용하므로 가능한 한 스핀락의 사용을 피해야 한다. 그러나 스핀락은 어떤 목적에서는 최상의 선택이다. 예를 들면, 만약 드라이버에게 지시되지 않은 패킷 수에 대한 참조 카운터들을 다루는 동안 드라이버가 스핀락을 요구한다면, 인터락드 연산을 사용하는 것은 불필요하다.

락킹 메커니즘에 대한 다른 제안들은 다음을 포함한다.:

* · 자원 풀들(예를 들면, NdisInitializeSListHead, NdisInterlockedPushEntrySList, NdisInterlockedPopEntrySList, NdisQueryDepthSList)을 다루는 것에 대해서는 Slists와 NDIS 단일 링크드 리스트 함수들을 사용해라.
* · 만약 스핀락을 사용한다면, 데이터는 보호하고 코드는 보호하지 않게 확실히 해라. 일반적인 경로들에서 사용되는 모든 데이터를 보호하기위해서 하나의 락을 사용하지 마라. 예를 들면, 보내기 경로가 그것의 데이터를 락할 필요가 있을 때 수신 경로가 영향을 받지 않게 하기위해서 두개의 데이터 구조체들에서 보내기와 받기 경로들에 사용되는 데이터를 분리해라.
* · 만약 스핀락을 사용하고 이미 경로가 DPC 레벨에 있다면, 여분의 코드를 피하기 위해서 DPC 레벨에서 스핀락 요구와 해제에 대한 함수들을 사용해라.(예를 들면, NdisDprAcquireSpinlock, NdisDprReleaseSpinlock)
* · 스핀락 요구와 해제의 수를 최소화하는 것에 대한 노력에서, NDIS ReadWriteLock 함수들 (NdisInitializeReadWriteLock, NdisAcquireReadWriteLock, NdisReleaseReadWriteLock)를 사용해라. ReadWriteLock 함수들은 여러 개를 동시에 읽는 자게 하나의 락을 사용하는 것을 허용하고 하나의 쓰는 자의 스레드에게 쓰기 접근을 제한한다. 쓰는 접근동안 읽기 접근이 허용되지 않는다.

**64-Bit DMA**  
만약 네트워크 어댑터가 64비트 DMA를 지원한다면, steps은 4기가 범위보다 상위의 주소에 대해 여분의 복사를 피해야 한다. 드라이버가 NdisMInitializeScatterGatherDma을 호출할 때, Dma64Addresses는 TRUE로 셋 되어야한다.

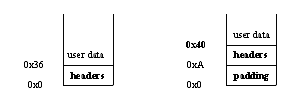
**Mapping Sent User Data**  
플래그는 초기화동안 NdisMSetAttributesEx에서 보낸 AttributeFlags 인자들을 추가한다. AttributeFlags에서 NDIS\_ATTRIBUTE\_USES\_SAFE\_BUFFER\_APIS 비트는 드라이버들이 전송된 사용자 데이터를 맵핑하는 것을 피하게 한다. 그때 드라이버는 데이터가 보내는 중일 때 NDIS로부터 보내진 물리 주소만을 필요로 해야 한다. 이 비트는 드라이버가 물리 주소만을 필요로 한다거나 드라이버가 NdisXxxSafe API 셋(밑에 있다)을 사용할 것인지를 NDIS에게 지시한다. Safe API를 사용할 때, 만약 NdisQueryBufferSafe각 버퍼 길이를 얻기 위해서 사용된다면, 가상 주소 인자는 NDIS에의해서 보내진 데이터 버퍼들을 매핑하는 것을 피하기 위해서 NULL로 셋 되어야 한다.

NDIS\_ATTRIBUTE\_USES\_SAFE\_BUFFER\_APIS을 셋팅하는 것은 가능할 때마다 시스템 가상 주소들에 버퍼들을 맵핑하는 것을 피하므로서 성능을 개선한다. 다음은 NdisXxxSafe API 셋이다.:

*NdisQueryBufferSafe*  
*NdisGetFirstBufferFromPacketSafe*   
*NdisCopyFromPacketToPacketSafe*   
*NdisBufferVirtualAddressSafe*

**Alignment**

캐쉬 라인 바운더리에서 버퍼 정렬은 하나의 버퍼에서 또 다른 버퍼로 데이터를 복사할 때 성능을 개선한다. 대부분의 네트워크 어댑터 수신 버퍼들은 처음 할당될 때 적당히 정렬된다 그러나, 실제 응용프로그램 버퍼에서 이벤트적으로 복사될 사용자 데이터는 소비된 헤더 공간에 의해서 정렬이 안된다. TCP 데이터(가장 일반적인 시나리오)에서 TCP, IP,이더넷 헤더들로 인한 쉬프트는 0x36바이트의 쉬프트의 결과를 낳는다. 일반적이 경우에 대한 이 문제를 해결하기위해서, 약간 더 큰 버퍼가 할당되고 패킷 데이터는 0xa 바이트 오프셋에 넣어야 한다. 이것은 헤더에 대한 0x36 바이트에 의한 버퍼의 쉬프트 후에 사용자 데이터가 적당히 정렬될 것이라는 것을 보증할 것이다.



**Scatter-Gather DMA   
[Back to Top](http://www.htsns.com/LoginTest/msnet/msdoc/high%20performance%20network%20adapters%20and%20drivers%20in%20windows.html#top)**

Scatter-Gather DMA 는 하드웨어에서 불연속적인 범위의 물리 메모리와의 데이터 송수신을 지원한다. Scatter-Gather DMA 는 SCATTER\_GATHER\_LIST 구조체를 사용하고 이 구조체는 SCATTER\_GATHER\_ELEMENTS 의 배열과 배열 안에 많은 엘리먼트들을 포함한다. 이 구조체는 드라이버의 보내기 함수에게 보내진 패킷 디스크립터로부터 회복된다. 배열의 각각의 엘리먼트는 물리적이고 연속적인 Scatter-Gather 영역의 시작하는 물리 주소와 길이를 제공한다. 드라이버는 데이터를 전송하는 것에 대한 주소와 길이를 사용한다.

DMA 연산에 대해서 Scatter-Gather 루틴을 사용하는 것은 만약 맵 레지스터들이 사용된다면 발생하는 정적으로 이 자원들을 락킹하는 것을 하지 않으므로 시스템 자원들의 이용률을 개선할 수 있다.

만약 네트워크 어댑터가 TCP 세그먼테이션 오프로드(큰 보내기 오프로드)을 지원한다면, 그때 드라이버는 최고 버퍼 크기로 보낼 필요가 있을 것이고 최고 버퍼 크기는 NdisMInitializeScatterGatherDma 안에 있는 MaximumPhysicalMapping 인자 안에 TCP/IP로부터 얻을 수 있다. 이것은 Scatter Gather 목록을 설립하고 가능한 버퍼 할당과 복사를 없앨 수 있는 충분한 맵 레지스터들을 가진다는 것을 보증할 수 있다.

**Auto-tuning   
[Back to Top](http://www.htsns.com/LoginTest/msnet/msdoc/high%20performance%20network%20adapters%20and%20drivers%20in%20windows.html#top)**

Auto-tuning는 드라이버 또는 네트워크 어댑터 하드웨어에서 수행될 필요가 있고 중요한 개념이다. 모든 불필요하고 불명확한 사용자 제어들을 제거하는 동안 최적의 성능을 유지하는 것은 중요하다.

최적의 성능 연산 포인트를 보증하기위해서, 네트워크 어댑터 드라이버는 Auto-tuning 인자들을 지원해야 한다. 드라이버는 주기적으로 드라이버 리셋없이 처리량과 응답 시간을 최고로 하기위해서 하드웨어와 소프트웨어를 조정한다. 모든 인자들이 조정되는 것은 아니고 특별히 하드웨어 인자들만 조정된다. Auto-tuning는 정적이 인자인게 더 좋다. 이유는 인자들의 최적의 값은 시간 시 변화할지도 모르기 때문이고 정적인 인자들은 사용자가 이해하고 올바르게 조정하기가 어렵기 때문이다.

Auto-tuning는 해결하기위해 어려운 문제이다. 많은 연구와 개발 시간이 초적의 조정 알고리즘을 이루는데 소비될 것이다. 그러나, 고려될 수 있는 많은 간단한 알고리즘들이 있고 auto-tuning과 가까운 최적의 연산 포인트을 합당히 이루는 것을 허용한다.

두 종류의 auto-tuning이 있다.: 정적, 동적. 정적 auto-tuning에서 드라이버와 네트워크 어댑터 하드웨어 인자들은 머신 역할(데스크탑에 대한 서버)과 머신 하드웨어(CPU와 메모리) 같은 시스템 구성과 네트워크 어댑터 하드웨어 능력들에 기반이 있다. 동적 auto-tuning는 자원 이용률과 네트워크 부하 같은 시스템 조건에 기반이 있다. 동적 auto-tuning는 수행하는데 더 어렵다. 그러나, 그것은 정적 auto-tuning보다 더 좋은 결과를 낳는다. 그러나, 정적 auto-tuning는 high-end 머신에서 자원들을 과다 이용하는 경향이 있다. 이유는 자원 사용을 다듬는 메커니즘이 없기 때문이다.

각각의 드라이버는 드라이버 초기화에서 레지스트로부터 로드되는 테이블을 유지하는 하라고 추천한다. 그 테이블은 네트워크 이용률을 포함하는 CPU 이용률(NdisGetCurrentProcessorCounts)과 다른 요소들에 기반된 연산 인자들을 선택한다. 드라이버는 타이머(각각 1초나 5초마다)에 기반되거나 예를 들면, 어떤 카운터들에 기반된((초당 패킷 수 같은) CPU 이용률을 계산하므로 이 인자들을 업데이트 할 수 있다.

auto-tuning에 대해 고려될 필요가 있는 주요 인자들은 다음의 아이템들을 포함한다:

* · Interrupt Moderation: Off 이나 On, 그리고 수신과 송신에 대한 인터럽트 비율
* · DPC당 처리되는 패킷이나 DPC안에서 소비된 시간
* · 버퍼 할당, 특히 수신 버퍼들
* · 합쳐진것을 작게 조각내는 것 같은 다른 처리

다음 테이블은 예제로서 제공된다.:

|  |  |  |  |
| --- | --- | --- | --- |
| **CPU Utilization %** | **Receive Packets/ Interrupt** | **Send Packets/ Interrupt** | **Packets processed at DPC Level** |
| <=5 | 1 | 1 | 200 |
| 10 | 1 | 1 | 180 |
| ... | ... | ... | ... |
| 95 | 19 | 70 | 45 |
| 100 | 20 | 80 | 40 |

레지스트리로부터 테이블을 로드하는 하나의 이유는 드라이버에서 어렵게 코드된 값들을 피하는 것이다. 이것은 또한 몇몇 representative 작업 부하에 대해서 최적의 값들을 찾기 위해서 테이블 엔트리로 실험하는 것을 허용한다. 실제 테이블은 사용자 간섭을 최소화하기위해서 사용자 인터페이스에서 보다 레지스트리에 보여야 한다. 드라이버는 테이블이 제공된다면 레지스트리를 사용해야 한다.

추가적으로, 네트워크 트래픽에 기반에서, 드라이버가 네트워크 어댑터의 최고 부하를 다룰 수 있을 만큼 충분한 버퍼들을 갖는 것은 매우 중요한다. 직면하는 가장 중요한 성능 이슈의 하나는 특별한 수신 버퍼에 서 충분한 버퍼의 부족이다. 버퍼들의 부족은 드라이버가 패킷들을 드랍하게 한다. TCP경우에, 이것은 재전송이 발생한다.(10% 정도의 높은 패널티가 네트워크 인센티브 작업 부하에 보여진다.). 뷰의 성능 포인트로부터, 그것은 그것들을 낮게 할당하는 것 보다 버퍼들을 높게 할당하는 것이 일반적으로 더 좋다. auto-tuning으로, 드라이버는 동적으로 관례에 기반된 수신 버퍼들을 할당할 수 있고 해제할 수 있다. 드라이버는 항상 네트워크 트래픽의 폭발을 다룰 수 있는 넉넉한 버퍼들을 가질 필요가 있다.

만약 하드웨어가 보내기와 수신 버퍼들에 할당된 하드웨어 메모리를 동적으로 조정하는 것을 지원한다면, auto-tuning는 보내기와 수신 패킷 률에 기반된 보내기와 수신 버퍼들을 위해 예비로 남겨둔 on-board 메모리를 조정할 수 있다. 이것은 on-board의 더 좋은 이용률을 허용한다.

**Call to Action   
[Back to Top](http://www.htsns.com/LoginTest/msnet/msdoc/high%20performance%20network%20adapters%20and%20drivers%20in%20windows.html#top)**

* · 너의 네트워크 어댑터와 드라이버가 최적의 시스템 성능 포인터에서 동작하는 것을 보증하기위해서 이 기사에 제공된 정보를 사용해라.
* · 현재 Windows DDK 는 이 기사에 설명된 API와 메커니즘들을 사용하는 법에 대한 자세한 정보를 포함한다.
* · 윈도우 XP를 실행하는 PC에서 빠른 시스템 시작을 위한 최적화에 대한 관계된 툴과 정보에 대해서는 OnNow and Power Management를 보아라.
* · 질문에 대해서, NDISFB@microsoft.com에 이메일을 보내라. 너의 이름,타이틀, 회사 이름, 회사 타입(IHV, ISV, ISP, or OEM)과 전화 번호를 포함하는 것을 확실히 해라.