Black Hat 2006
Reinventing TCP/IP in Windows Vista with the NetIO stack

Abolade Gbadegesin

Architect

Contact: aboladeg at microsoft dot com

Getting Started

About me

Responsible for architecture of network transports in Windows

11 years working on Windows networking

6 years redesigning the Windows networking stack

What this talk will cover

Guiding principles

NetIO architecture

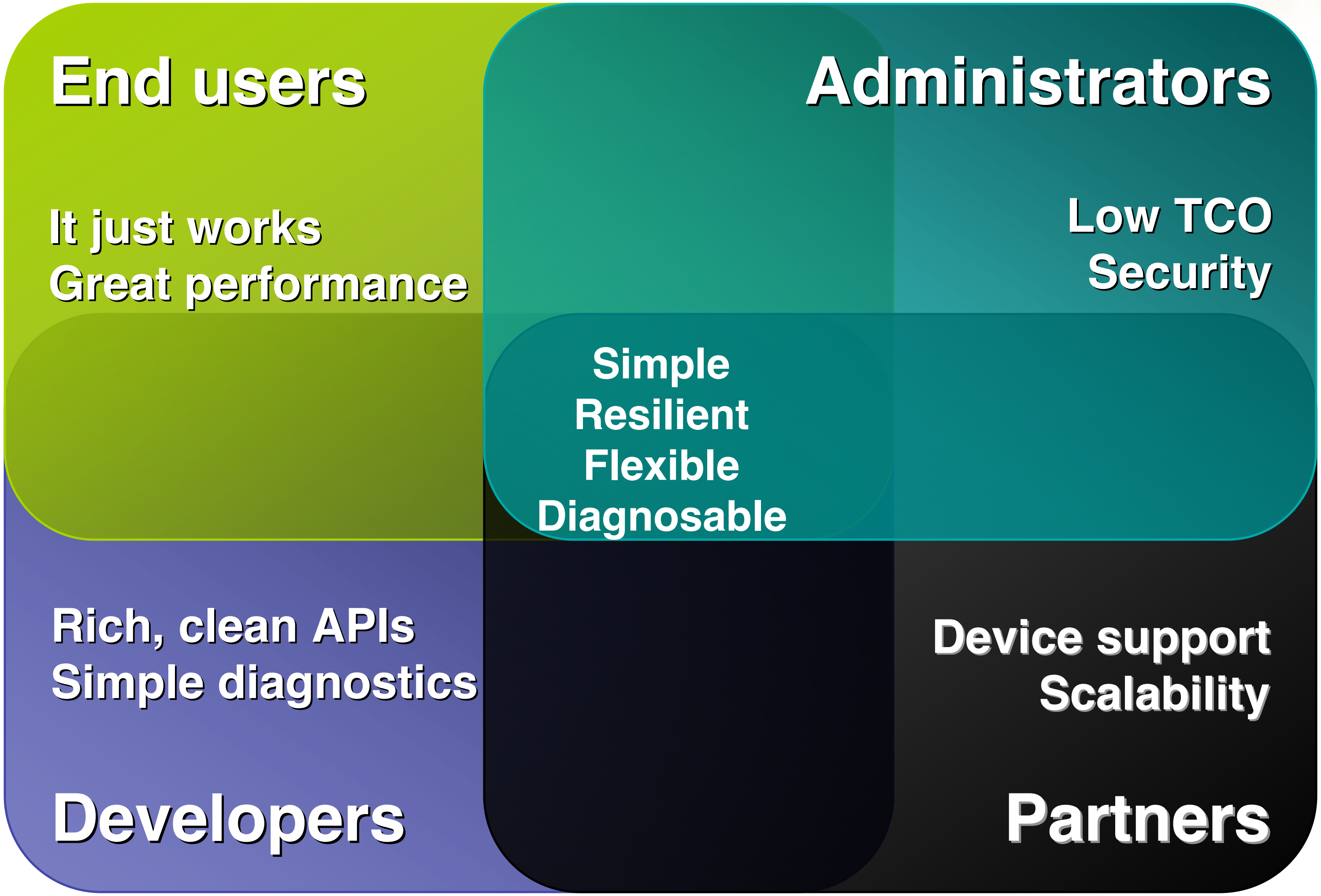
Integrated and extensible network security

Performance and scalability

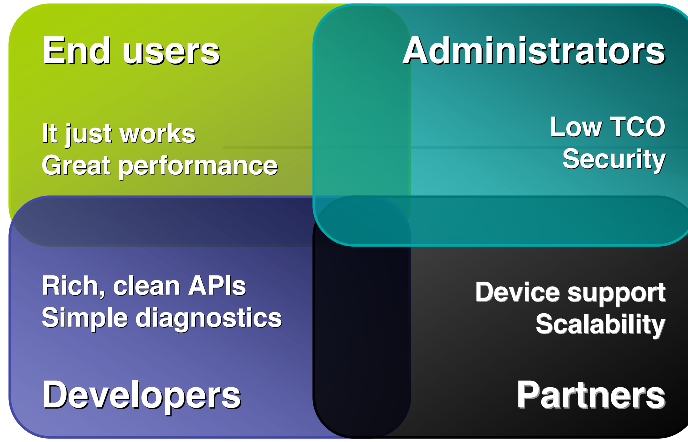
Writing networked applications

Reinventing TCP/IP

What do customers want?

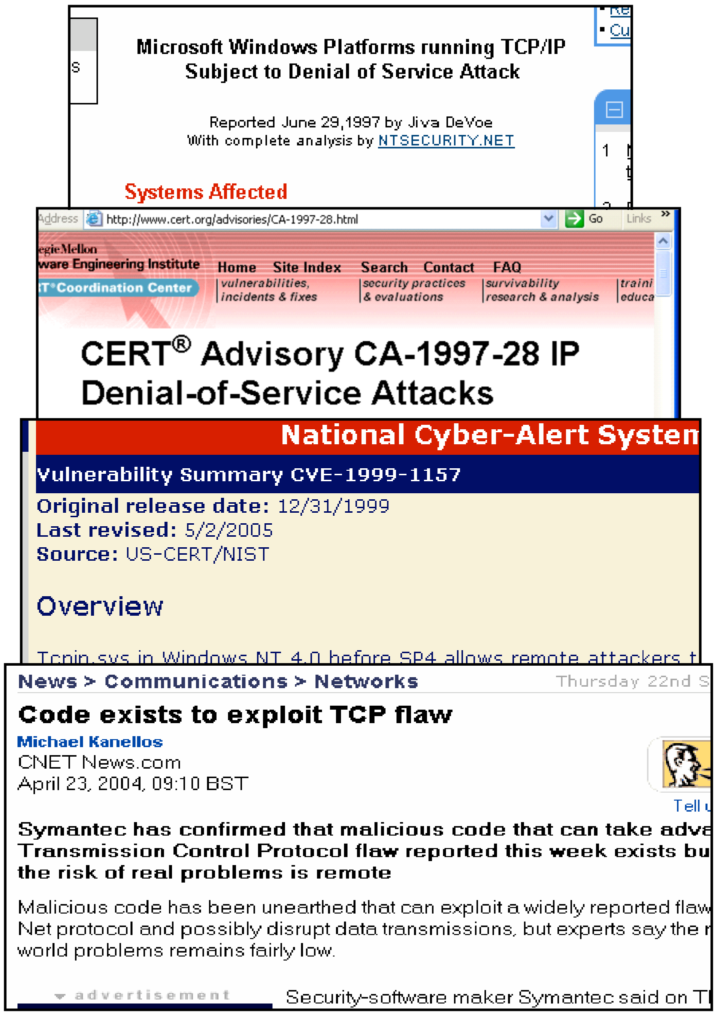


Reinventing TCP/IP
Guiding Principles



- Define the **state of the art** in networking
- Design components to be **extensible and diagnosable**
- Raise the bar on **security and resilience**
- Enable pervasively **flexible and self-tuning** performance

History



- Denial-of-service exploits
  - Early attacks exploited spoofing, protocol design and product vulnerabilities
  - Current attacks use stateful, non-spoofed sessions from 0wned machines
- Penetration exploits
  - Code-execution vulnerabilities more rare in low layers like TCP
  - Attacks moving farther up the application stack!

Timeline

Late 2003
3rd milestone
NetIO TCP/IP in winmain builds

Dual-family sockets
Multicast
Early performance work

Mid 2002
2nd milestone
1st TCP data exchange
Early connection offload work
IPv4 and IPv6 fragmentation and reassembly

Incoming TCP connection setup over IPv6

IPv6 neighbor discovery
Outgoing TCP connection setup over IPv6

Early 2001
1st milestone
UDP sockets over IPv6

BVT framework running

Project checkins begin

Initial architecture and design decisions

1999

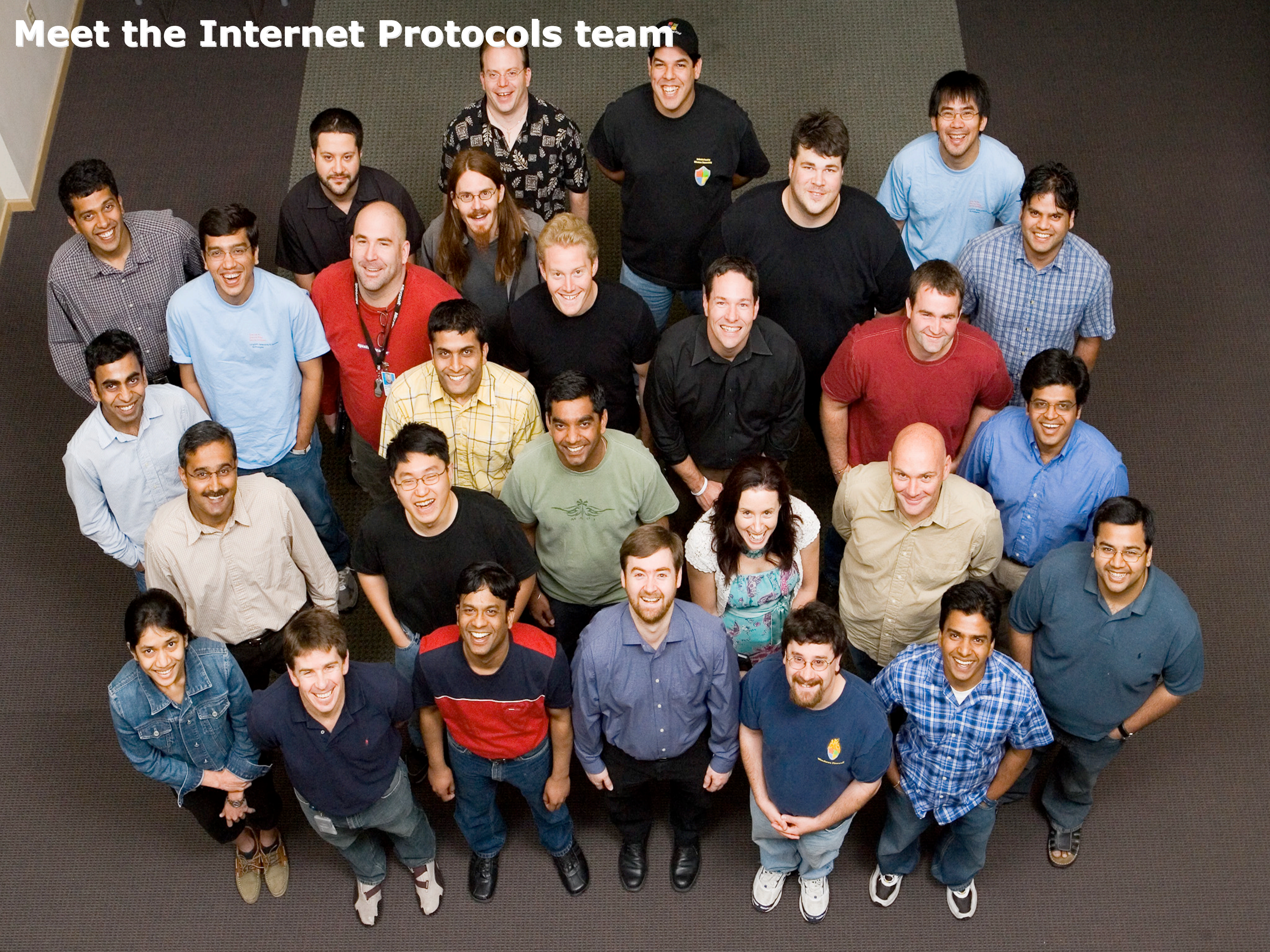
2000

2001

2002

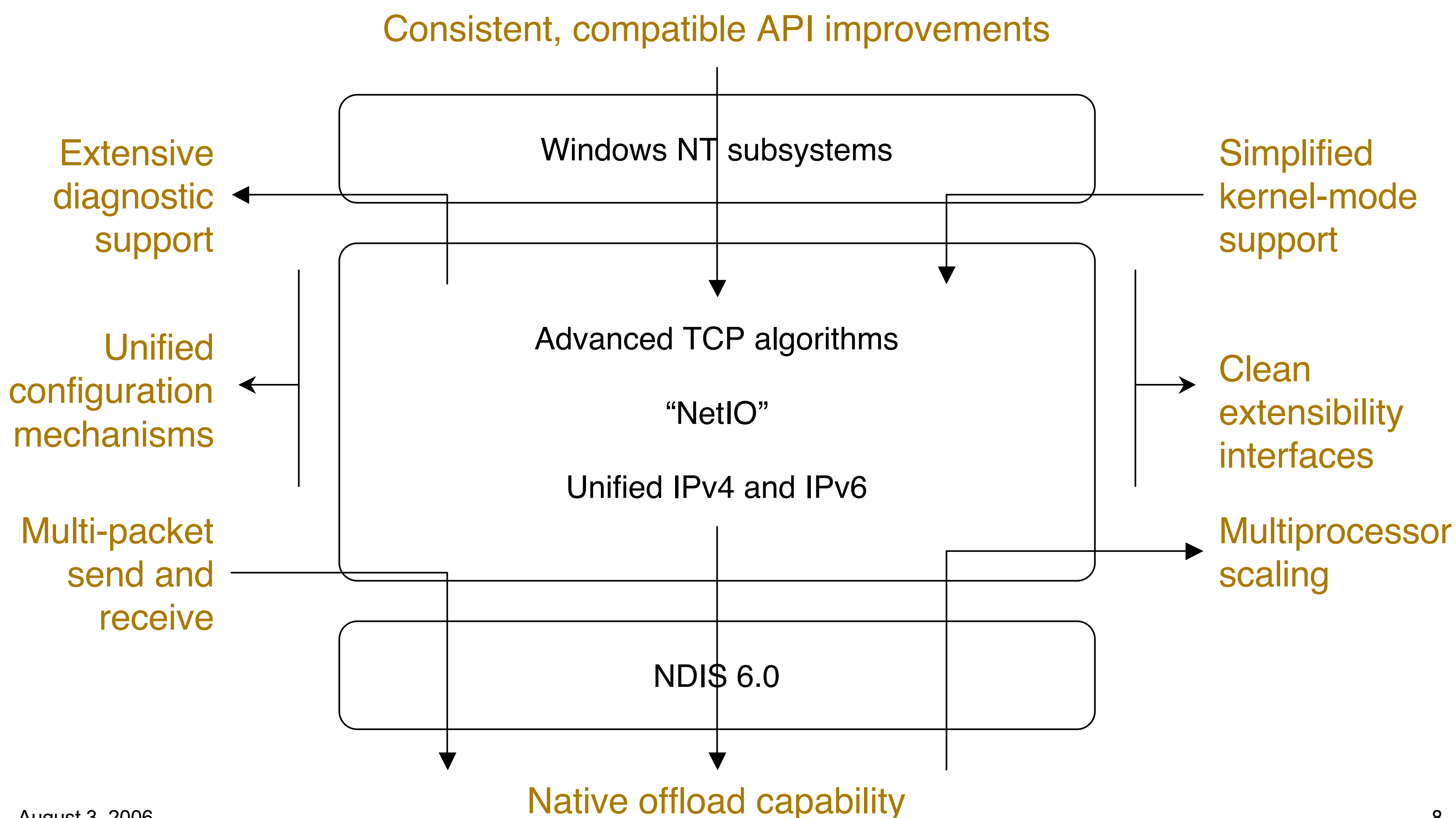
2003

Meet the Internet Protocols team



NetIO architectural framework

Goals and design decisions



NetIO architectural framework

A whirlwind debugger-guided tour

- Putting the components together
  - modules, binding, configuration
  - transport and network protocols
  - diagnostics, tracing
- Maintaining runtime state
  - compartments, interfaces, addresses, routes
  - endpoints, ports, listeners, connections
- Handling I/O
  - requests, buffers, queuing
  - paths, neighbors
  - inspection, injection, callouts

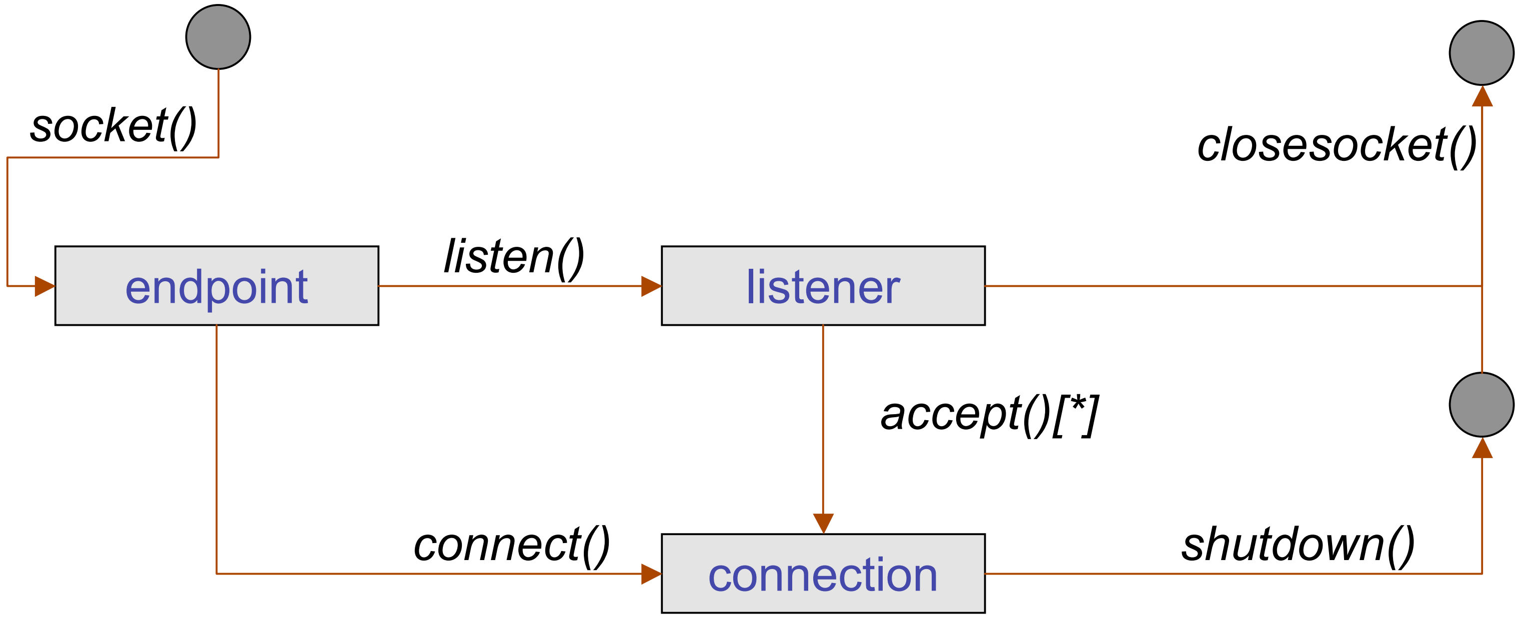
Designing for extensible security

One question, though…

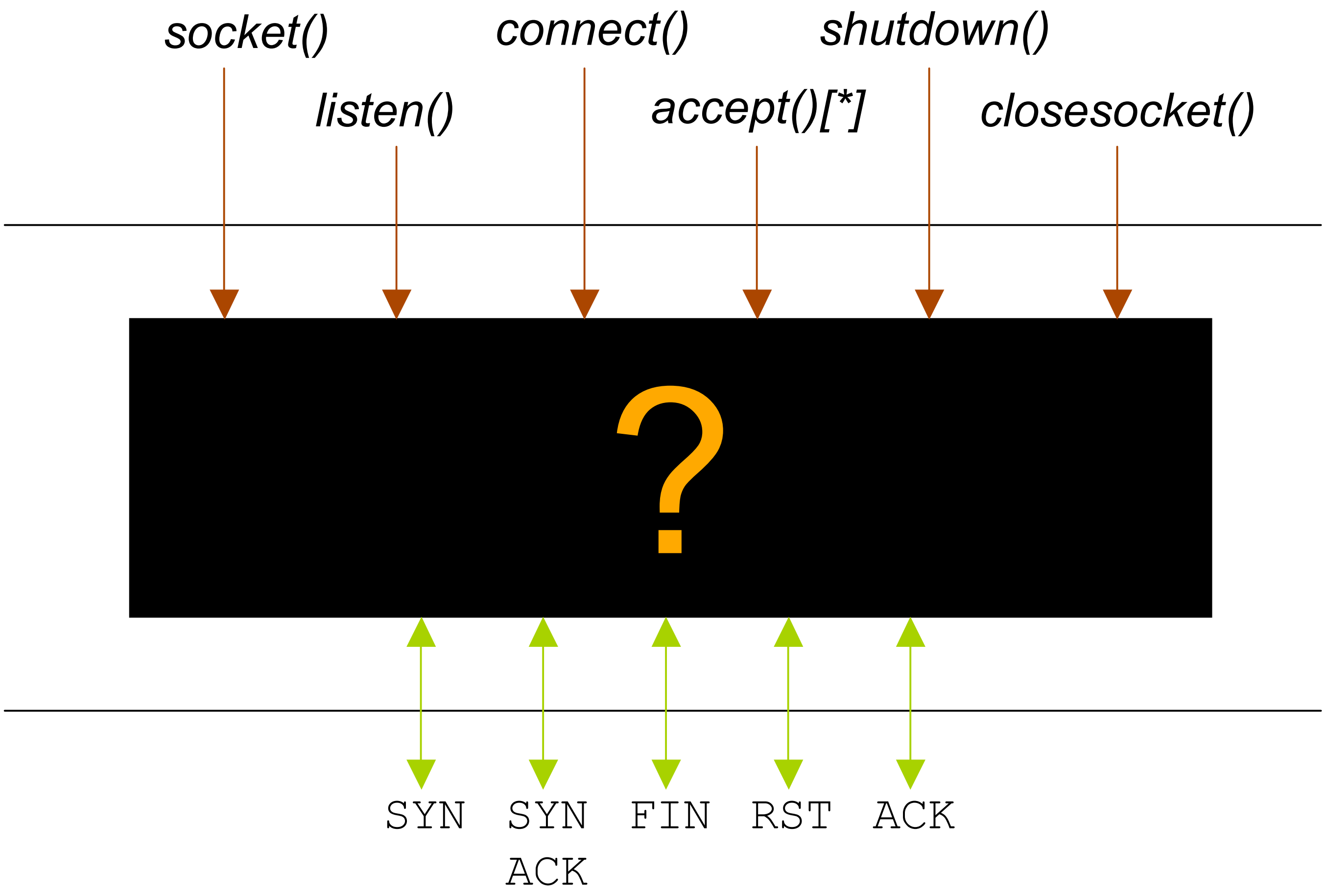
What does all this change mean for network security and network policy solutions on Windows?

Designing for extensible security

The problem: how do you infer this…



Designing for extensible security
...from this?



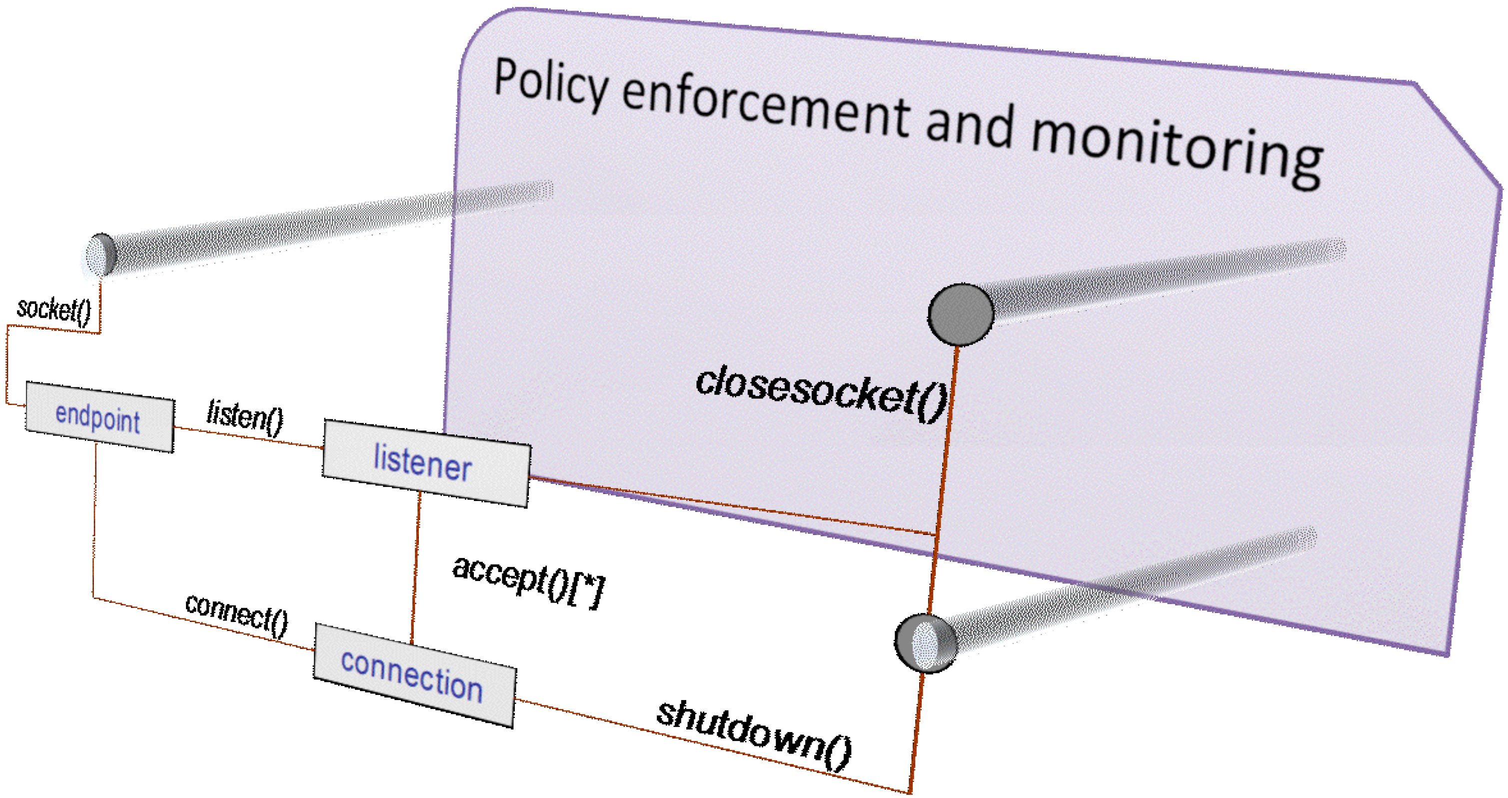
Designing for extensible security

Three conclusions

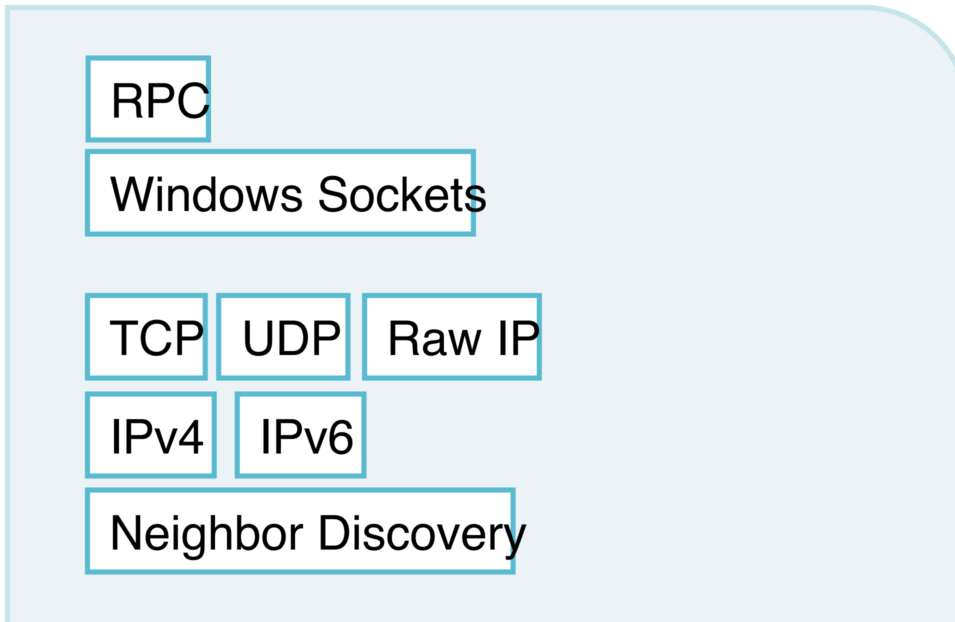
- Security-focused components need visibility into the operation of the things that they secure
- Policy-enforcing components need direct control over the things that policy talks about
- Security policy must be decoupled from components so it can evolve at the pace of security threats

Designing for extensible security
The NetIO approach

Allow external components to cleanly observe and influence internal logic

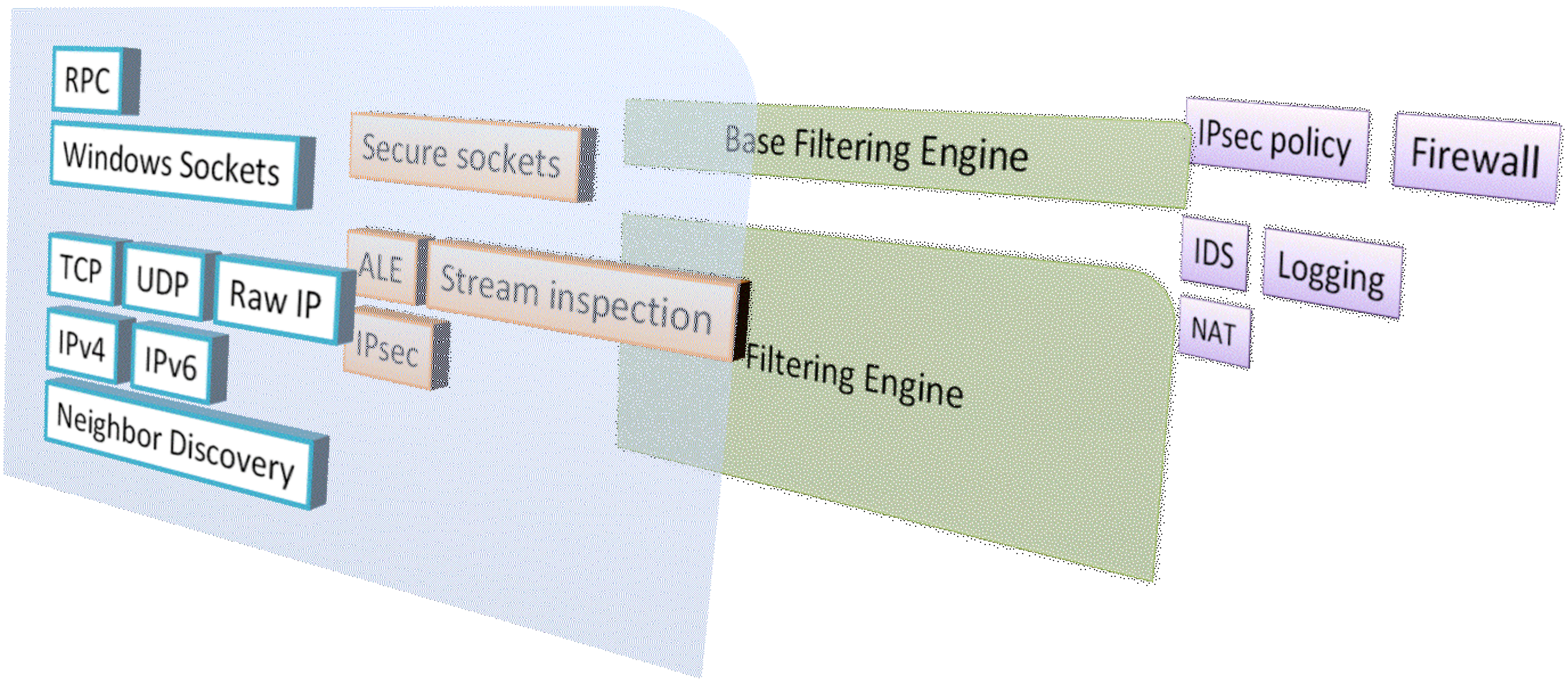


This is the networking stack…



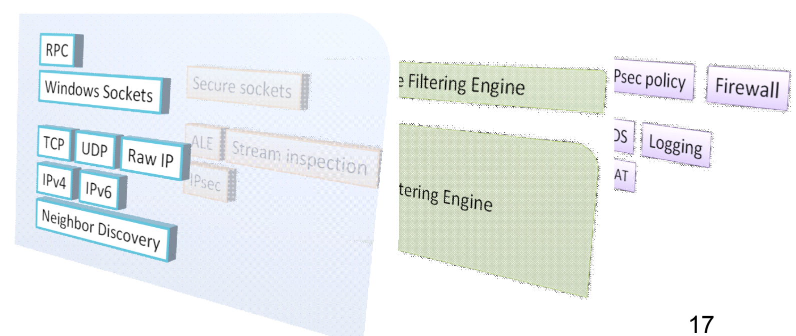
Designing for extensible security
Understanding the Windows Filtering Platform

…and this is how WFP fits in.



Designing for extensible security
What's in the picture?

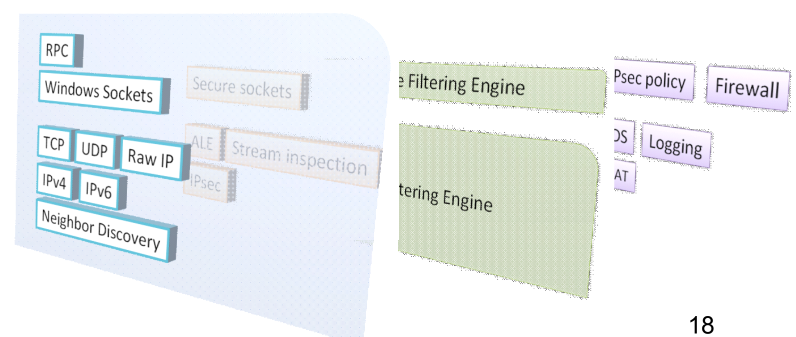
- Core stack (TCP, UDP, IPv4, IPv6)
- Built-in policy-related components
  - Application Layer Enforcement
  - Stream inspection
  - IPsec
- Core filtering engine
  - User-mode and kernel-mode logic
  - Filter database
- Filtering callouts



Designing for extensible security

What layers are defined for callouts?

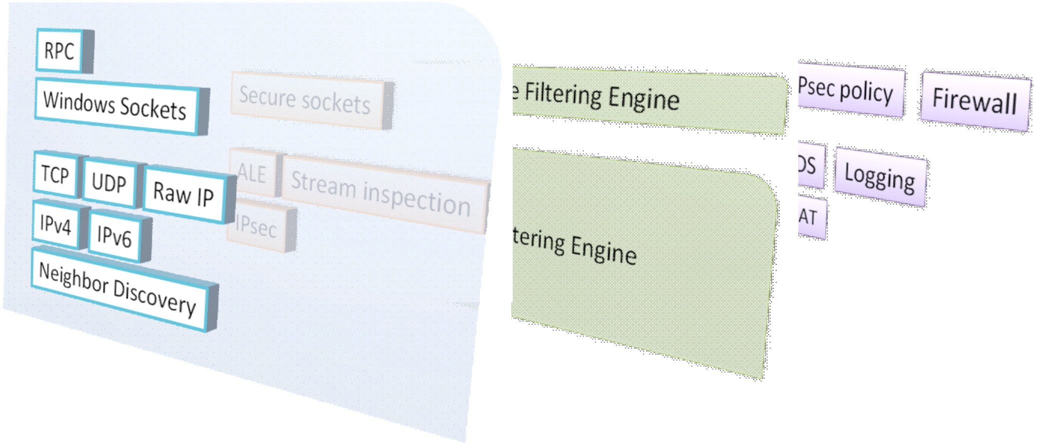
- RPC, IKE
- Socket operations (listen, accept, connect, port assignment)
- In-order TCP data streams
- Inbound & outbound TCP/UDP messages
- Inbound, outbound & forwarded IP packets
- ICMP messages
- …and more to come



Designing for extensible security

What does WFP enable?

- Extensibility
- Transparency to users and applications
- Tight integration, high performance, scalability



Performance and scalability

Core TCP performance

Handling intensive workloads

Core TCP performance

Flow control 101

Receiver advertises window

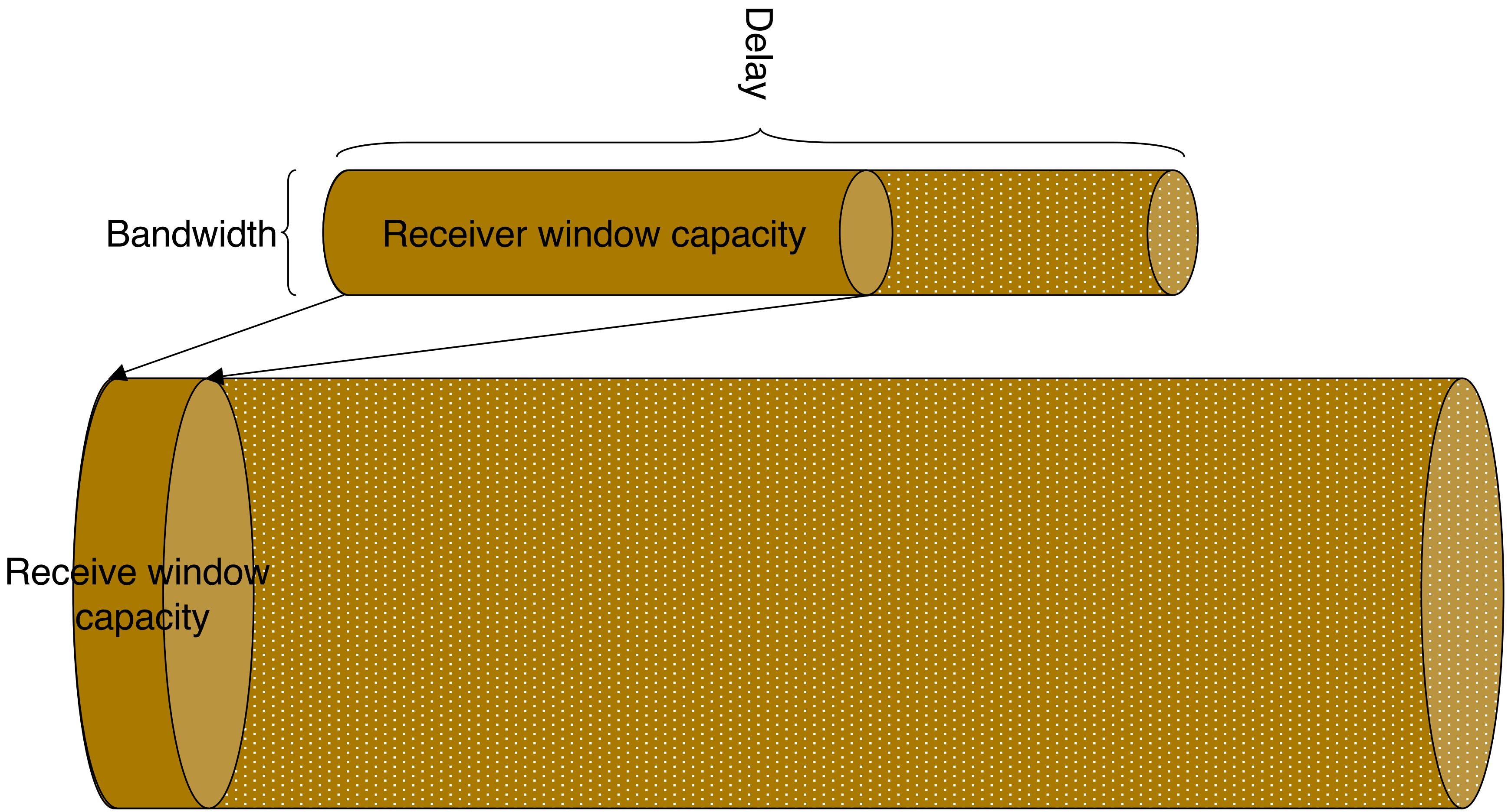
Sender transmits up to window size

Receiver has to acknowledge something before sender can transmit further

Ideal window: bandwidth * delay

Core TCP performance

Pipes & flow control



Core TCP performance

Flow control on various paths

Pipe characteristics	Ideal window	Capacity utilized by default window
100 Mbps with 10 ms delay	128KB	~12%
5Mbps with 200ms delay	128KB	~12%
1Gbps with 50ms delay	~6MB	~1.2%

Core TCP performance

TCP receive window auto-tuning

Receiver enables window scaling by default

Continuously estimates pipe capacity and monitors application reads

Auto-tunes receive window advertisements to ensure the receive window doesn’t limit throughput

up to 4000% improvement over XP in throughput for HTTP

up to 4600% improvement over XP in throughput for file transfers with SMB 2.0 pipelining

Core TCP performance

Controlling auto-tuning

Command line:

netsh interface tcp set global autotuninglevel

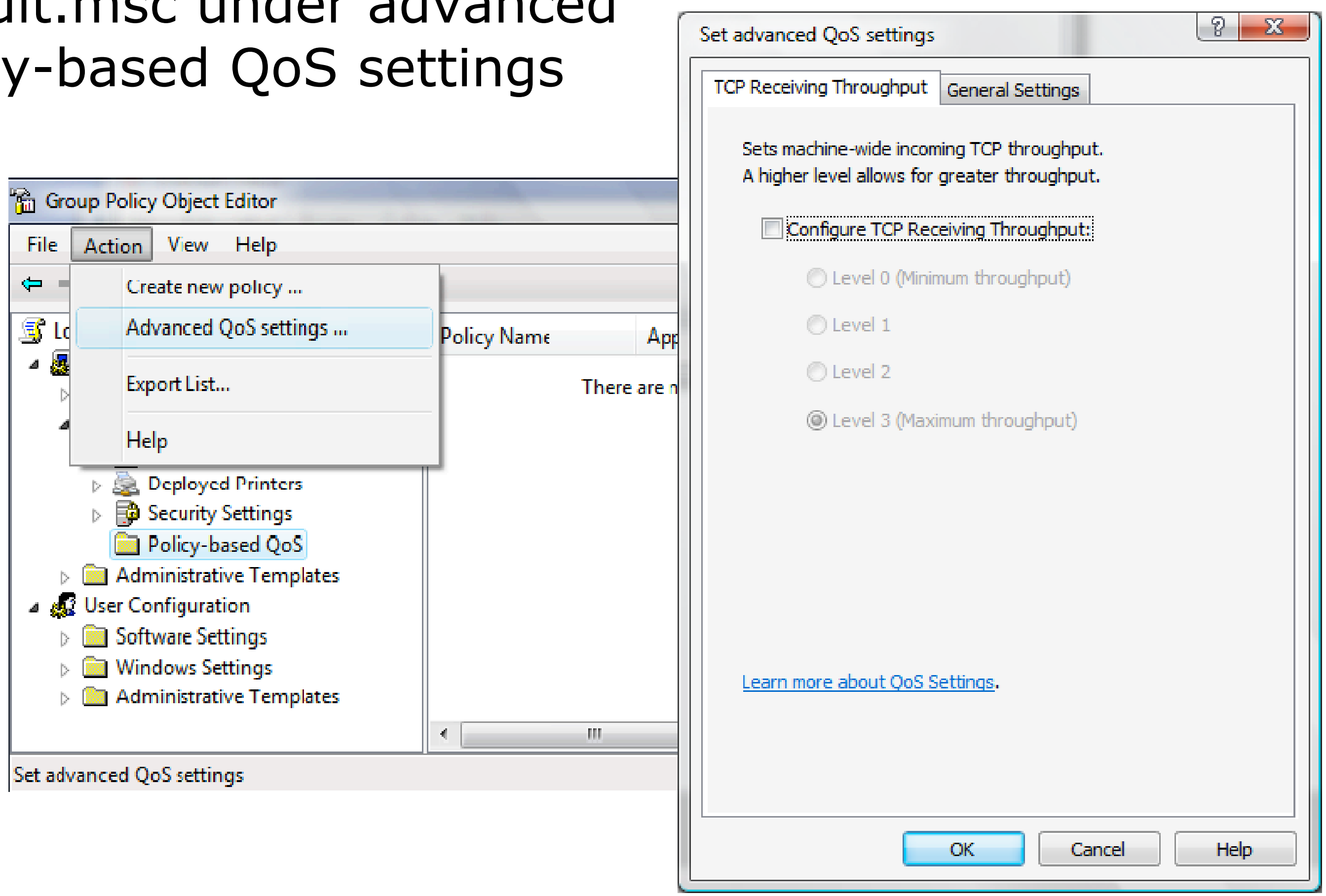
```
D:\Users\aboladeg>netsh interface tcp set global help

Usage: set global
             [[autotuninglevel=]
                disabled|highlyrestricted|restricted|normal|experimental]

Parameters:

       Tag             Value
       autotuninglevel - One of the following values:
                         disabled: Fix the receive window at its default
                             value.
                         highlyrestricted: Allow the receive window to
                             grow beyond its default value, but do so
                             very conservatively.
                         restricted: Allow the receive window to grow
                             beyond its default value, but limit such
                             growth in some scenarios.
                         normal: Allow the receive window to grow to
                             accomodate almost all scenarios.
                         experimental: Allow the receive window to grow
                             to accomodate extreme scenarios.
                             WARNING: This can dramatically degrade
                             performance in common scenarios and should
                             only be used for research purposes.
```

Core TCP performance

Controlling auto-tuning

Group Policy and UI

Gpedit.msc under advanced policy-based QoS settings



Performance and scalability

Core TCP performance

Handling intensive workloads

Handling intensive workloads

Tackling bandwidth scalability

Speed	Budget
1 Gbps	16 _secs
10 Gbps	1.6 _secs
100 Gbps	0.16 _secs

Minimizing per-packet processing

Multi-packet transmission and reception

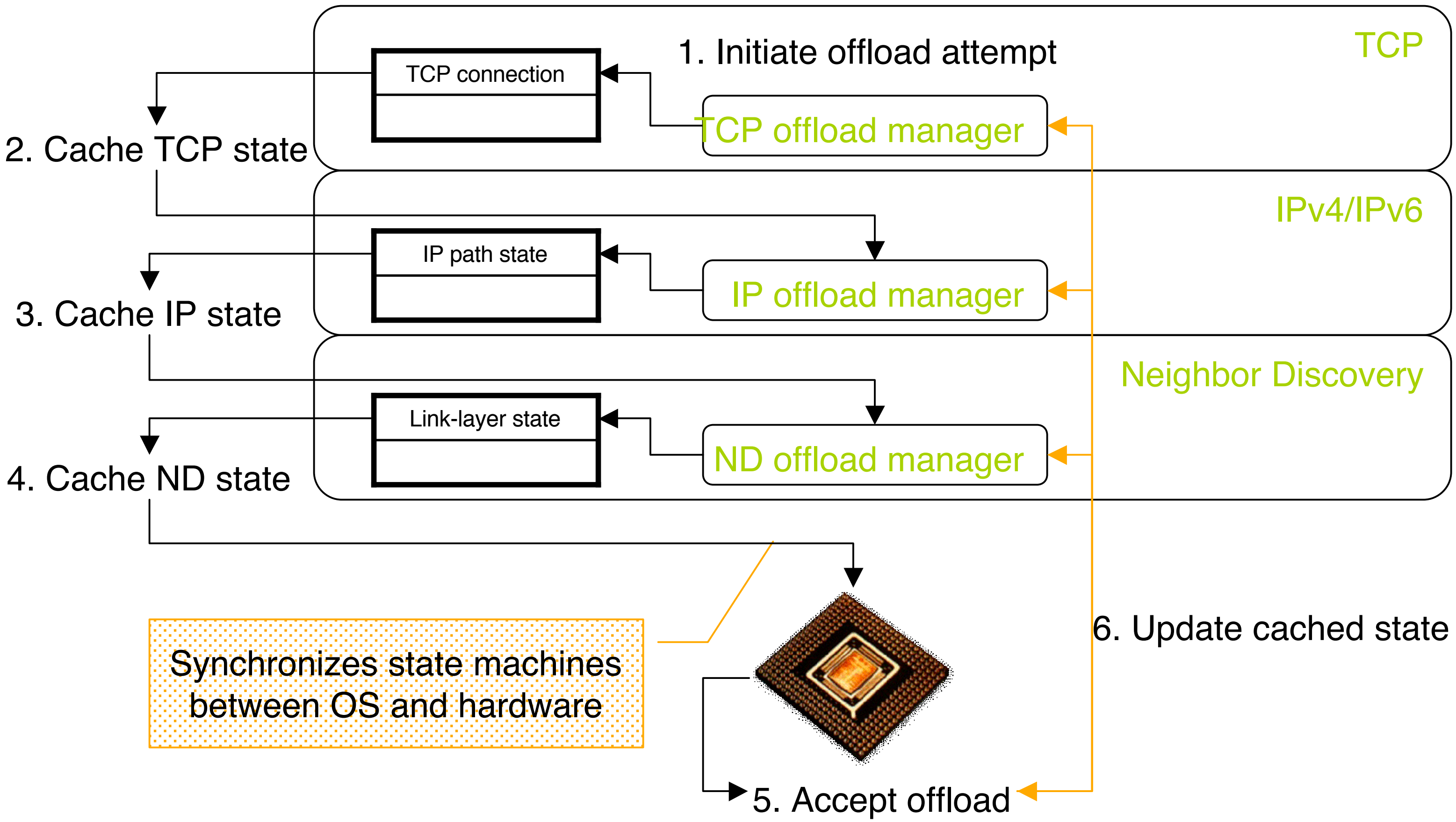
Offload checksum computation and verification

Giant Send Offload (GSO)

but driving a connection at 100 Gbps requires more….

Handling intensive workloads

TCP connection offload



Handling intensive workloads

TCP connection offload

Transparently and gracefully transitions state back and forth between OS and hardware

Defines offload state composably to simplify offload of other protocol stacks (e.g. SSL)

OS continuously monitors connection activity and selects suitable candidates for offload

greater than 50% reduction in CPU utilization using 1Gbps Ethernet for HTTP workloads

Handling intensive workloads
High latency throughput

Path characteristics	Buffer size	Packets in flight
1 Gbps at 500ms	~64MB	~32 thousand
10Gbps at 500ms	~512MB	~256 thousand
100Gbps at 500ms	~6GB	~3 million

- Packet loss probability grows steadily
- Ramp-up after loss takes much longer (10 minutes on 1Gbps/100ms path)

Handling intensive workloads

Classic TCP congestion control 101

Slow-start phase

Increase congestion window by 1 packet for each cumulative acknowledgment

Congestion avoidance phase

Increase congestion window by 1 packet for each round trip

Congestion response

On loss, drop window to 1 packet and set slow-start threshold to _ outstanding data

Handling intensive workloads

Delay-based TCP congestion control 101

Congestion avoidance

Detect congestion by sensing increased delay

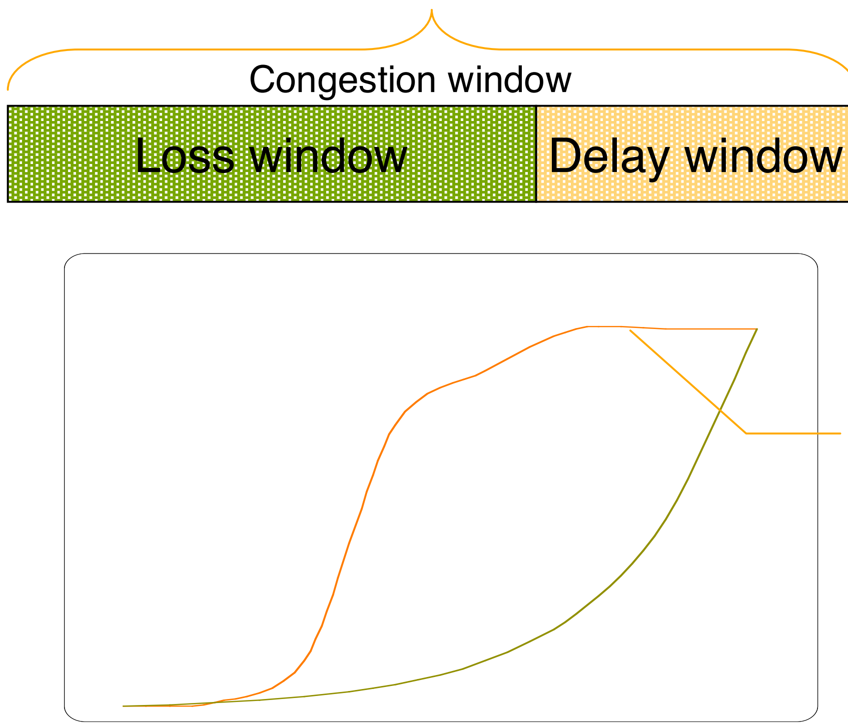
Assumes sufficient network buffering to produce measurable delay variations

Congestion response

Avoid packet loss by adjusting congestion window in response to delay

Handling intensive workloads

Reducing ramp-up time with Compound TCP



Combine loss and delay windows for faster ramp-up and recovery

Handling intensive workloads
Compound TCP

Tries to avoid losses when running alone and recover quickly from losses caused by others

Designed for fairness to connections using loss-based congestion control

nearly 50% reduction in transfer time over 1Gbps path with 30ms RTT

Writing Networked Applications

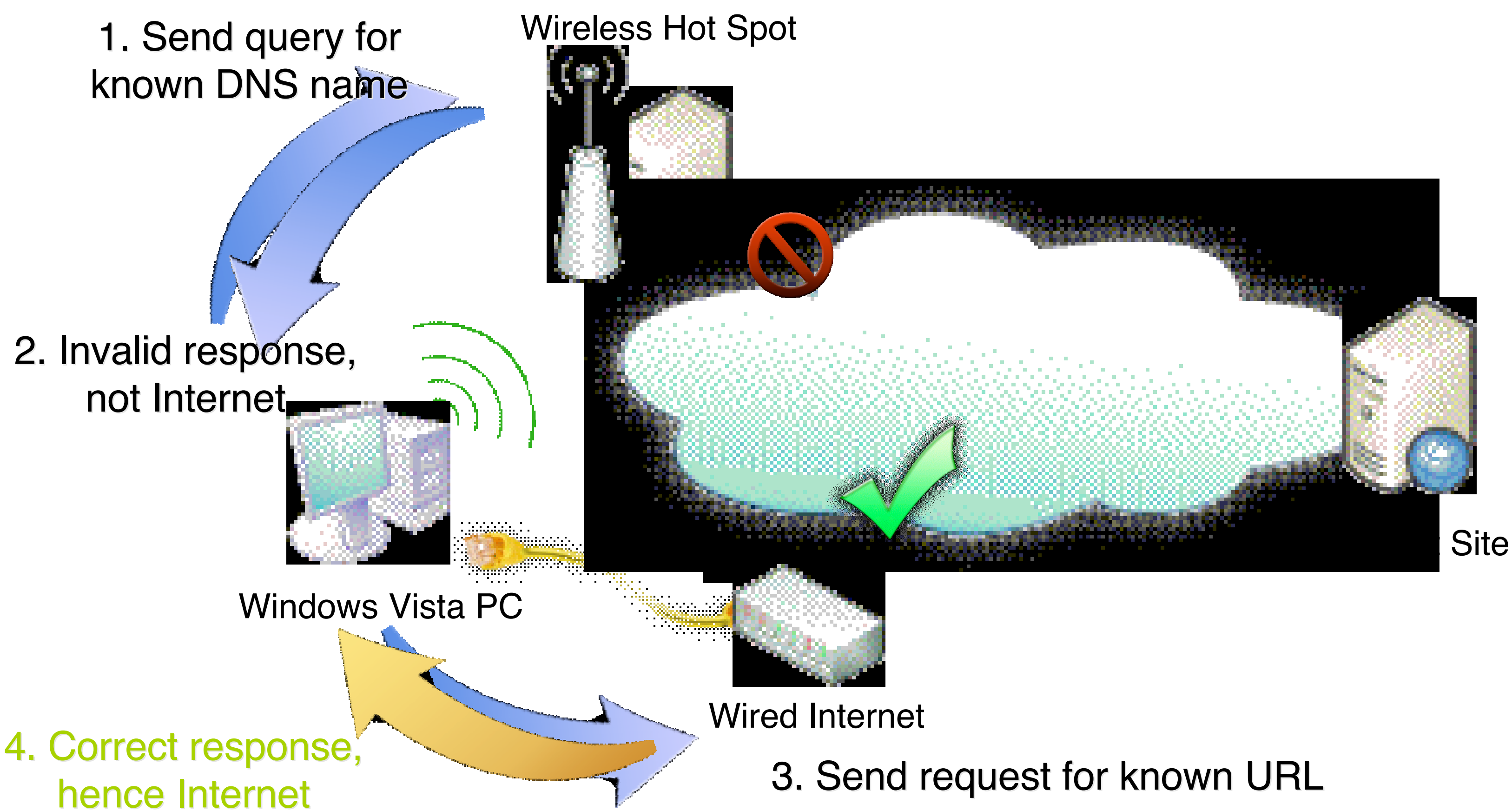
Detecting Internet connectivity

Optimizing connection establishment

Port management

Writing networked applications

Detecting Internet connectivity



Writing networked applications

Detecting Internet connectivity

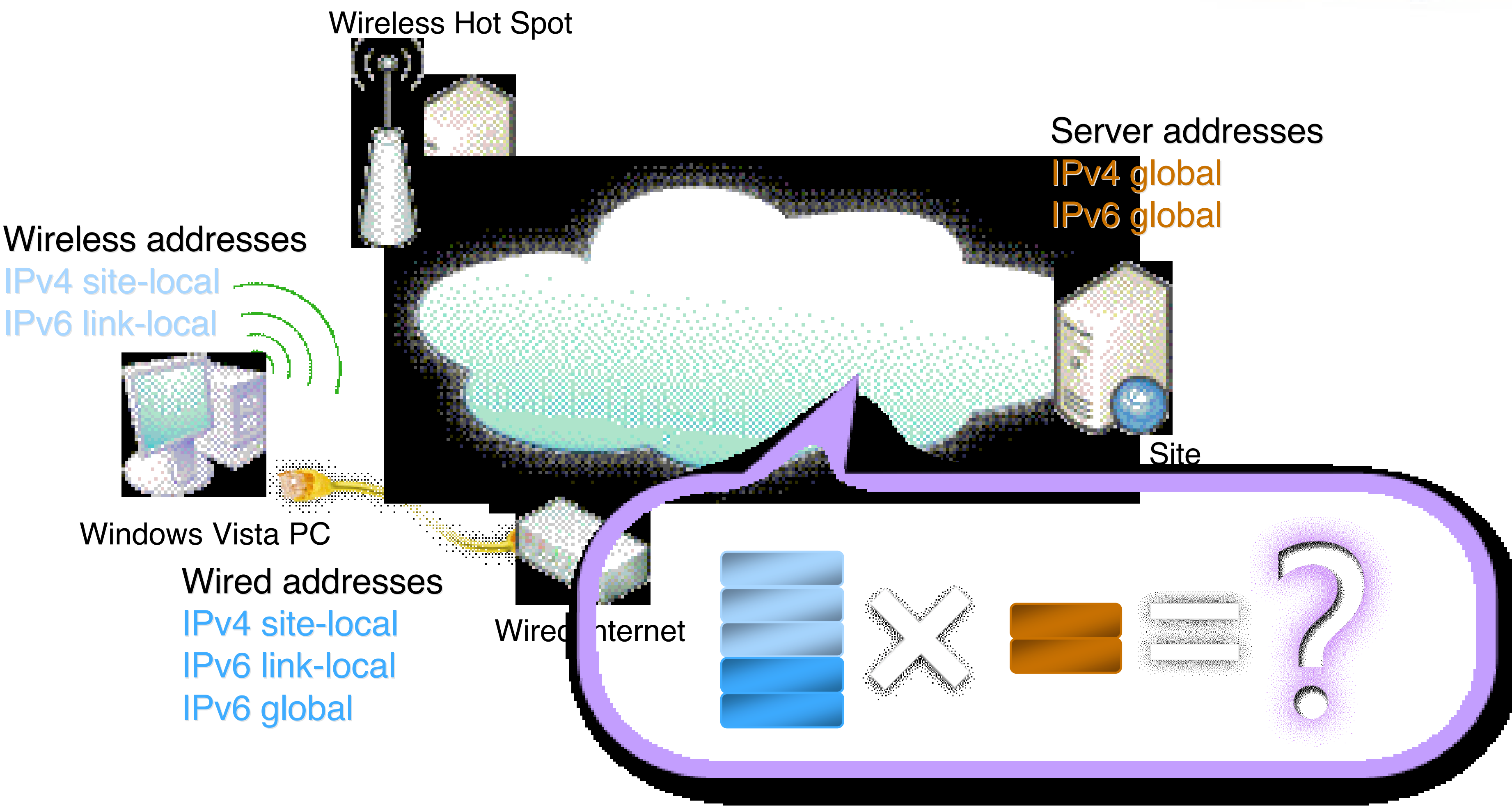
Queries issued through Network Location Awareness API, handled by NLA 2.0 service

Handles DNS spoofing by wireless hotspots and detects transparent HTTP proxies

Scales by leveraging DNS and HTTP caching

characterizes global Internet connectivity for both IPv4 and IPv6

Writing networked applications
Optimizing connection establishment



Writing networked applications

WSAConnectByName and WSAConnectByList

Prioritizes and sorts multiple combinations of source and destination addresses

Currently tries one combination at a time, will make parallel attempts in future

Address sorting functionality available on its own via socket I/O control

designed to optimize connection success rate across IPv4 and IPv6

Writing networked applications

Basic port management

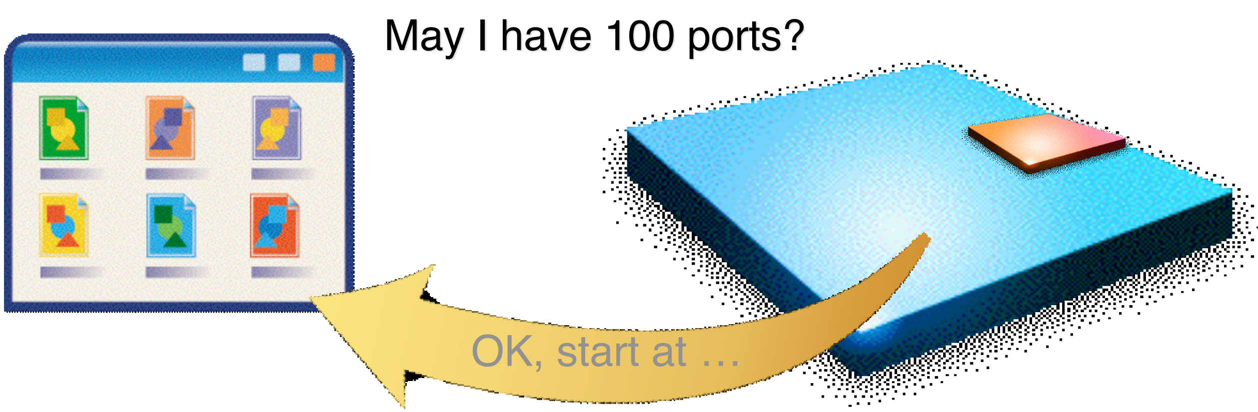
Windows Server 2003	1… Well-known ports …1024	1025… Ephemeral ports …5000	5001… Other ports …65534
Windows Vista	1… Well-known ports …1023	1024… Registered ports …49151	49152… Ephemeral ports …65534

more port numbers for dynamic assignment

fewer collisions on registered port numbers

Writing networked applications

Reserving ports at runtime for applications



Writing networked applications
Reserving ports statically for services

Writing networked applications

Port management

Supports IANA compliance for registered and ephemeral ports

Reserve port numbers at runtime and statically

Optionally randomizes port assignments for increased security

Call to Action

We're building the foundation, and we want your help!

- Ensure your tools & products light up with NetIO
  - Test devices for compatibility with TCP window scaling
  - Achieve great TCP performance by supporting pipelining and multithreading
  - Extend your reach by supporting IPv4 and IPv6
  - Leverage new features, e.g. port reservation and randomization

Call to Action (2)

We're building the foundation, and we want your help!

- Innovate on NetIO to enable new scenarios
  - Plug into WFP to enforce your own security policies
  - Use secure sockets for authentication & authorization
  - Leverage kernel sockets & kernel IP helper API in drivers

Resources

- Email
  - TCP/IP: tcpipfb@microsoft.com
  - WFP: wfp@microsoft.com
- Windows Vista on MSDN and TechNet
  - http://msdn.microsoft.com/windowsvista/
  - http://windowssdk.msdn.microsoft.com/
  - http://www.microsoft.com/technet/windowsvista/network/default.mspx
- The Cable Guy
  - http://www.microsoft.com/technet/community/columns/cableguy/default.mspx

Still to Come!

13:45 – 15:00	WiFi in Windows Vista: A Peek Inside the Kimono	Noel Anderson & Taroon Mandhana
15:15 – 16:30	Windows Vista Heap Management Enhancements – Security, Reliability and Performance	Adrian Marinescu
16:45 – 18:30	Case Study: The Security Development Lifecycle and Internet Explorer 7	Tony Chor

secure@microsoft.com