Implementing IPv6 for Windows NT

Richard P. Draves, Microsoft Research
Allison Mankin, USC/ISI
Brian D. Zill, Microsoft Research
Main Points

• Windows NT is a good base for network protocol development
• Our release is great sample code: http://research.microsoft.com/msripv6
Outline

• Motivation
• Windows NT Networking
• Our Implementation
• Problems & Solutions
• Source Code Access
• Performance
• Conclusions
Motivation

- Primarily a learning experience
- Bootstrap Microsoft on IPv6
- Platform for further research
Windows NT Networking

**User Process**
- Application
  - Winsock
    - ws2_32.dll
      - Winsock 2
    - WSH
      - wship6.dll
        - Winsock Helper for IPv6
    - WSP
      - msafd.dll
        - MS’s Winsock Provider
      - rnr20.dll
        - MS’s Namespace Provider

**Existing Components**
- Added Components

**Kernel**
- afd.sys
  - Driver for Winsock
  - TDI
  - tcpip6.sys
    - IPv6 Protocol
  - ndis.sys
    - Device-Independent Driver
  - dc21x4.sys
    - Device-Specific Driver
  - ndis.sys
    - Device-Independent Driver
  - ndis.sys
    - Device-Independent Driver

- NDIS
  - NDIS
Our Implementation

- Started with NT 4.0 TCP/IP source code
- Supports only IPv6
- Supports only NT 4.0/5.0
Our Implementation

**Upper-Layer Protocols**
- TCP
- UDP
- ICMP
- MLD

**Core**
- Neighbor Discovery
- Send/Receive
- Routing

**Link-Layer Modules**
- LAN
- Loopback
- Tunnel
Problems & Solutions

- NDIS receive handlers
- Adding link-layer headers
- “Pull-up” non-contiguous packet data
- Preventing deadlock with NDIS and TDI
NDIS Receive Handlers

- Asynchronous callbacks
- ProtocolReceive
  - flat look-ahead buffer
  - may need separate call to transfer data
- ProtocolReceivePacket
  - NDIS_PACKET structure with buffer chain
  - not implemented by all NICs
  - miniport owns the packet/buffers
NIC Implementations

• `ProtocolReceive + transfer data`
  – Intel EtherExpress 16

• `ProtocolReceive`
  – SMS EtherPower II
  – 3com Fast Etherlink XL
  – Intel EtherExpress PRO

• `ProtocolReceivePacket`
  – Digital DE435
Our NDIS Receive Handler

• Link-layer module hides complexity
  – Pass up our own IPv6_PACKET structure
• Supports both receive handlers
  – IPv4 code only supported ProtocolReceive
• Does transfer-data internally if needed
  – May introduce a copy relative to IPv4
Adding Link-Layer Headers

- Must construct link-layer header before handing packet to NDIS
- NT 4.0 IPv4 code chains a buffer in the link-layer module
  - Adds complexity
  - Reduces performance
- Allocate space up front
  - But how much space?
  - NDIS does not support a packet offset
Our Solution

- Leave room for worse-case link-level header
- Rewrite NDIS packet to hide unused space
  - Must undo this after the send completes
  - Communicate offset value in the context area
  - What if the unused space spans two pages?
Source Code Access

• Source for Windows NT 4.0 TCP/IP
  – Sample code, UDP/TCP, TDI glue
  – Replaced all link-layer, IP, ICMP, MLD code
  – DDK sample code

• Source for other Windows NT components
  – Not essential
  – Useful for debugging & documentation
  – Our only modification was a fix in msafd.dll
## Performance

TCP Throughput in KB/s

<table>
<thead>
<tr>
<th></th>
<th>10 Mb/s</th>
<th>100 Mb/s</th>
</tr>
</thead>
<tbody>
<tr>
<td>IPv4</td>
<td>1058±4</td>
<td>10995±20</td>
</tr>
<tr>
<td>IPv6</td>
<td>1032±3</td>
<td>10790±30</td>
</tr>
</tbody>
</table>

- Expected 1.4% slower, saw 2.5% / 1.9%
- 300Mhz P-II -> 266Mhz, SMC Etherpower II
Conclusions

• Windows NT is a good base for network protocol development

• Our release is great sample code: http://research.microsoft.com/msripv6
  – Testing, research, educational uses