



# Stratus Technologies

---

**Uptime.** All the time.

## OpenVOS Network Application Performance

---

**Noah Davids**  
Stratus Customer Assistance Center

January 18, 2012



# OpenVOS Network Application Performance



- Network Properties
- TCP Stack
- Application Design
- Network Emulation Tools

# Network Properties



# Network Properties



- Latency
- Loss/Corruption
- Bandwidth

# Network Properties - Latency



- Definition
  - The time it takes from the transmission of the first bit of a packet until the arrival of the last bit of the packet
  - There are other definitions, I think this makes the most sense since until the last bit is received you cannot process the packet

# Network Properties - Latency



- Minimum latency will be based on the distance and path
  - Fiber/copper cables
    - ~ 70% speed of light - 4.76 microseconds/kilometer
    - 19.6 milliseconds between San Francisco and New York
  - Satellite in geostationary orbit (22,236 miles)
    - 100% speed of light - 3.34 microseconds/kilometer
    - 239 millisecond
    - Up and back down
  - Satellite in Medium Earth (1420 Km) or Low Earth (670 Km) orbit
    - ~ 40 milliseconds (up and down)
    - Medium and Low Earth satellite links have much lower throughput than geostationary orbit satellites
- **Network elements will add to latency**

# Network Properties - Latency



- Ping is a reasonable estimate BUT
  - ICMP echo request/reply packets may be blocked
  - May be given a lower priority
  - May not reflect TCP or UDP travel times
  - Network acceleration devices may reduce apparent latency for TCP but not ICMP
  - Application will probably have a different latency than ping
  - Measures round trip time – roughly 2x the latency BUT that assumes latency is symmetrical

# Network Properties - Latency



- Phoenix AZ to Maynard MA
  - 2,666.3 miles (4,291 kilometers) (a very rough estimate)
  - Expected latency 20 ms
  - Round trip time should be a minimum of 40 ms

# Network Properties - Latency



```
ping 134.111.200.229 -count 10
```

```
Pinging host 134.111.200.229 : 134.111.200.229
ICMP Echo Reply:TTL 53:TOS is 0x6 time = 73 ms
ICMP Echo Reply:TTL 53:TOS is 0x6 time = 72 ms
ICMP Echo Reply:TTL 53:TOS is 0x6 time = 71 ms
ICMP Echo Reply:TTL 53:TOS is 0x6 time = 71 ms
ICMP Echo Reply:TTL 53:TOS is 0x6 time = 71 ms
ICMP Echo Reply:TTL 53:TOS is 0x6 time = 71 ms
ICMP Echo Reply:TTL 53:TOS is 0x6 time = 71 ms
ICMP Echo Reply:TTL 53:TOS is 0x6 time = 71 ms
ICMP Echo Reply:TTL 53:TOS is 0x6 time = 78 ms
ICMP Echo Reply:TTL 53:TOS is 0x6 time = 79 ms
ICMP Echo Reply:TTL 53:TOS is 0x6 time = 82 ms
Host 134.111.200.229 replied to all 10 of the 10 pings
ready 12:05:52
```

# Network Properties - Latency



```
sc 6000 164.152.77.217 100 1
```

```
sc 6000 164.152.77.217 100 1
```

Transaction completed in 72 milliseconds

Transaction completed in 72 milliseconds

Transaction completed in 72 milliseconds

Transaction completed in 72 milliseconds

Transaction completed in 72 milliseconds

Transaction completed in 72 milliseconds

Transaction completed in 72 milliseconds

Transaction completed in 72 milliseconds

Transaction completed in 73 milliseconds

Transaction completed in 72 milliseconds

ready 15:34:34

- SC is a program that sends messages of X bytes using 1 or 2 calls to the send function.
- The SC server echoes the last 11 bytes of the application message so it must wait for the entire message to arrive.
- Time is measured from when the send call returns until the echoed message is received
- Here 100 bytes are sent using 1 call to the send function.

# Network Properties - Latency



- Latency becomes much more important when TCP and application turns are considered.
  - Turns are many exchanges must be performed before a result is determined
- About the only thing you can do about latency is
  - Change the routing so it is more direct
  - Change the routing so there are fewer network elements
  - Replace the current network elements with faster elements
  - In most cases latency is like the weather, you can plan for it but you can't do anything about it
    - Not planning can be fatal to an application

# Network Properties – Loss/Corruption



- From the TCP perspective there is no difference. Both result in dropped packets requiring retransmissions
  - However correcting the problems require different solution sets
- TCP sender will significantly throttle back how much data it sends when a packet is dropped.
  - Congestion control
- TCP sender will send missing packet after getting 3 duplicate ACKs without waiting for the retransmission timer to expire
  - fast retransmit

# Network Properties - Loss/Corruption



- Transaction based applications, like SC, will not be harmed by congestion control or helped by the fast retransmit.
- Not sending enough packets at one time to be affected by throttling
- Not sending enough packets at one time to get three duplicate ACKs
  - Would need to send at least 4 MSS bytes worth of data
    - MSS – maximum segment size – maximum number of bytes in a TCP segment
- What hurts is the retransmission timeout – how long it takes resend the packet

# Network Properties - Loss/Corruption



sc 6000 164.152.77.128 100 1

sc 6000 164.152.77.128 100 1

Transaction completed in 2 milliseconds

Transaction completed in 1 milliseconds

Transaction completed in 1 milliseconds

Transaction completed in 1 milliseconds

Transaction completed in 1 milliseconds

Transaction completed in 1 milliseconds

Transaction completed in 1 milliseconds

Transaction completed in 1 milliseconds

Transaction completed in 1 milliseconds

Transaction completed in 263 milliseconds << dropped packet had to be retransmitted

ready 14:22:39

# Network Properties - Loss/Corruption



## ■ Applications that send a lot of data at one time are impacted by throttling

### • 0% packet loss

```
ftp> put simple_client4.pm x
200 PORT command successful.
150 Opening data connection for x (164.152.77.217,52009)0.
226 Transfer complete.
1155072 bytes sent in 8.59 seconds (131.39 Kbytes/sec)
```

### • 1% packet loss

```
ftp> put simple_client4.pm x
200 PORT command successful.
150 Opening data connection for x (164.152.77.217,52017)0.
226 Transfer complete.
1155072 bytes sent in 25.73 seconds (43.85 Kbytes/sec)
```

### • 0% packet loss (sanity check)

```
ftp> put simple_client4.pm x
200 PORT command successful.
150 Opening data connection for x (164.152.77.217,52023)0.
226 Transfer complete.
1155072 bytes sent in 11.19 seconds (100.85 Kbytes/sec)
```

# Network Properties - Loss/Corruption



- What can you do about loss
  - Usually happens in the network
    - Network element queues overflow
    - Very hard/impossible to figure out without the cooperation of network people
    - Document the loss with simultaneous traces on both client and server networks then complain to the people responsible for your network support
  - But it could be local – see netstat slide

# Network Properties - Loss/Corruption



- What can you do about corruption
  - See the previous slide, “What can you do about loss”
  - But could also be local to end points
    - Typically caused by duplex mismatch
    - Could also be bad cabling

# Network Properties - Loss/Corruption



```
netstat -interface #sdlmux.m17.10-2.0.11-2.0
```

```
Ethernet adapters are grouped  
Number of failovers = 0
```

```
Active Device Statistics:
```

```
MAC Type      : CSMA/CD  
MAC Address: 00:00:a8:42:ba:64  
Device Name: #sdlmux.m17.10-2.0.11-2.0
```

```
Line Speed : 100 mb/s  
Line Duplex: Full-Duplex
```

```
MAC Statistics:
```

```
. . . .  
  Transmit frame discarded, late collisions: 0  
. . . .  
  Transmit frame discarded, excessive retry: 0  
  Receive frame discarded, lack of buffers : 0  
  Receive frame discarded, improper framing: 0  
  Receive frame discarded, an overflow      : 0  
  Receive frame discarded, bad CRC         : 0  
  Receive frame discarded, bad address     : 0  
  Receive frame discarded, congestion      : 0
```

# Network Properties - Loss/Corruption



- Duplex mismatch
  - One side set for auto-negotiate the other set for full duplex and not doing auto-negotiation
    - Side doing the auto-negotiation must assume half duplex
    - Auto-negotiate/half duplex side will see increasing counters
      - Transmit frame discarded, late collisions
      - Transmit frame discarded, excessive retry
        - These are sources of loss, not corruption
    - Full duplex side will see increasing counters
      - Receive frame discarded, improper framing
      - Receive frame discarded, bad CRC
        - These are indications of incomplete frames

# Network Properties - Loss/Corruption



- Bad Cabling
  - Haven't see this in quite a while but still need to consider it if there is any indication of corruption (or loss) and it cannot be traced to a duplex mismatch
  - A bad cable could affect the packets from "server to switch" while leaving the "switch to server" transmission unaffected
    - Just because the server does not see any indication of corruption doesn't mean the switch doesn't

# Network Properties - Loss/Corruption



- Other indications of packet loss
  - Receive frame discarded, lack of buffers
  - Receive frame discarded, an overflow
  - Receive frame discarded, congestion
- I don't think I have ever seen the overflow or congestion counts  $> 0$
- Don't worry if you see lack of buffers  $> 0$ 
  - Unless you can correlate increasing counter to an actual problem
    - Should be investigated to prevent future problems
  - Most common reason is broadcast storms

# Network Properties - Bandwidth



- Bandwidth is the total capacity of the network – this is a theoretical maximum
- Throughput is the total application data capacity of a network
  - Need to subtract out the data link (Ethernet), network (IP) and transport (TCP or UDP) headers
    - Small application data frames have a corresponding larger overhead
  - Need to subtract out network housekeeping data (ARP, SNMP, other stuff)
  - Need to subtract out other applications
  - Need to factor in application limits

# Network Properties - Bandwidth



- Assuming you are not exceeding your bandwidth (and dropping packets because of congestion) increasing bandwidth will probably not gain you any increase in throughput.

# Network Properties - Bandwidth



- How to tell if you have run out of transmit bandwidth
  - Look at the switch to see if it is running out of buffers or reporting excessive queue length
  - Also check the "netstat -interface" statistics for "Transmit ring full"
    - This is not really a bandwidth issue but the effect is the same
      - the card's "bandwidth" has been exceeded

# Network Properties - Bandwidth



- How to tell if you have run out of receive bandwidth
  - Look at the switch to see if it is running out of buffers or reporting excessive queue length
  - Also check the "netstat -interface" statistics for "Receive frame discarded, lack of buffers" and "Receive frame discarded, congestion"
    - These are not really bandwidth issues but the effect is the same

# TCP Stack



# TCP Stack



- Maximum Segment Size (MSS)
- ACK delay
- Jumbo Frames
- Send/Receive windows

# TCP Stack – Maximum Segment Size (MSS)



- The maximum number of bytes that can be accepted in 1 TCP segment
- Can be no larger than the local Maximum Transmission Unit (MTU) minus headers
  - Typically 1460
  - Will be reduced if the host uses TCP options
- This is advertised
- Each side can have a different value
- Typically both sides use the smallest of the advertised values

# TCP Stack – Maximum Segment Size (MSS)



- STCP uses
  - 1460 for local communication
  - 536 for remote (going through a router, regardless of geography) communication

# TCP Stack – Maximum Segment Size (MSS)



- Why is it important to know what the MSS is
  - By default the system waits for an ACK when there is a packet smaller than the MSS value
    - This is configurable by the application
  - The result is that if you are sending data such that  $(\text{number of bytes mod MSS}) \neq 0$  you can double the transaction time.
    - It can be faster to pad the data length so that  $(\text{number of bytes mod MSS}) = 0$
  - This is known as Nagle's algorithm



# TCP Stack – Maximum Segment Size (MSS)



- Trying to pad to  $(\text{number of bytes mod MSS}) == 0$  can be a moving target
- TCP options can take up space so while 536 bytes will fit in talking to 1 host the may not fit talking to another and the application has no way tell

```
sc 6000 134.111.200.185 525 1
Transaction completed in 376 milliseconds
. . .
sc 6000 134.111.200.185 524 1
Transaction completed in 74 milliseconds
. . .
sc 6000 134.111.200.229 536 1
Transaction completed in 74 milliseconds
```

# TCP Stack – Maximum Segment Size (MSS)



- Controlled by the `min_mss` parameter
  - Minimum value is 576 (has added in the 20 byte IP and 20 byte TCP headers)
  - **Increasing MSS affects all connections may result in fragmentation or dropped packets if a link cannot support the larger frame size**

```
as: list_stcp_params min_mss -long
```

```
lowest IP max segment size [500-1480] (min_mss) 576
                                         default_min_mtu/4
```

# TCP Stack – Maximum Segment Size (MSS)



- Two benefits of increasing MSS to 1460
  - Assuming packets > 536 bytes
    - Fewer packets so less overhead
    - Faster response for packets between 536 and 1460 or whatever the MSS really is.

# TCP Stack – Maximum Segment Size (MSS)



```
analyze_system -request_line 'set_stcp_param min_mss 1480' -quit
. . . .
Changing maximum tcp segment size (min_mss)
  from 556 to 1480

sc 6000 164.152.77.217 1000 1
. . . .
Transaction completed in 359 milliseconds
Transaction completed in 358 milliseconds
```

# TCP Stack – Maximum Segment Size (MSS)



- Does not seem to have helped
- Remember that MSS is advertised. So if the remote doesn't specify a larger MSS we will use the MSS that it advertises.
- STCP will actually use the smaller of the local and the remote MSS so need to change both

```
sc 6000 164.152.77.217 1000 1
```

```
. . . .
```

```
Transaction completed in 76 milliseconds
```

```
Transaction completed in 75 milliseconds
```

# TCP Stack – ACK delay

- TCP has a minimum over head of 40 bytes
  - 20 bytes in the IP header
  - 20 bytes in the TCP header
    - Assuming there are no options
- An acknowledgement without data over an Ethernet segment is a minimum of 64 bytes
  - In an attempt to reduce overhead TCP will delay acknowledging some packets for up to 200 milliseconds in the hopes that the application will send some data and the ACK can be piggy backed onto the data
    - This is controlled by the `ack_delay` parameter
      - Minimum value is 40 ms
  - TCP will immediately ACK every second full sized (MSS bytes) packet so streaming data is not effected by this feature

# TCP Stack – ACK delay



```
sc 6000 164.152.77.217 1000 1
. . . .
Transaction completed in 359 milliseconds
Transaction completed in 364 milliseconds

[[ ack_delay set to 40 ms on the server ]]

sc 6000 164.152.77.217 1000 1
. . . .
Transaction completed in 155 milliseconds
Transaction completed in 158 milliseconds
```

# TCP Stack – Jumbo frames



- Supported starting in 17.1
- Can only be used on gigabit links
  - Can NOT be used over a router, MSS limited to 1480
  - All hardware in the network must support the larger MTU
    - Or the packets get dropped
- MAX size of frame depends on hardware

# TCP Stack – Jumbo frames



- 1500 bytes
  - Embedded adapters in V100, V200, V400
  - U571V Single-Port 10/100/1000 mbps copper adapter
- 9200 bytes
  - Embedded adapters in V150, V250, V300, V500
  - Embedded adapters in V2302, V2404, V4304, V4408, V6308, V6408
  - U574V-LC Dual-Port 1000 mbps fiber adapter
  - U575V Dual-Port 10/100/1000 mbps copper adapter
  - U578V Quad-Port 10/100/1000 mbps copper adapter
  - U582V Quad-Port 10/100/1000 mbps copper adapter
  - U583V Quad-Port 1000 mbps fiber adapter
  - U776V Quad-Port 1000 mbps fiber adapter
- 16110 bytes
  - U584V Dual-Port 10,000 mbps fiber adapter

# TCP Stack – Jumbo frames



- What will it buy you?
  - Latency should not be an issue on a gigabit network BUT
  - Putting all data in 1 frame instead of several may eliminate delayed ACK issues
    - Depends on the application
  - Less frame overhead
    - Assuming that the packets will contain more than 1460 bytes of data

# TCP Stack – Send/Receive windows



- Receive window
  - Advertised by the receiver
  - Indicates how much buffer space is left
    - Space to hold bytes that have not been read by the application
  - Sender can send a window's worth of bytes before having to stop and wait for an acknowledgment
    - Other conditions, like slow start and congestion control may further limit the number of bytes that the sender can send without waiting for an ACK
  - Receiver may advertise a 0 window if the application has not read any data

# TCP Stack – Send/Receive windows



- Send Window
  - How much data the sender can hold without getting an acknowledgment from the receiver
    - Acknowledged bytes have been received by the receiver's TCP stack not read by the receiving application
  - Sending application will block or get an EWOULDBLOCK error once the send window fills
    - The fact that the sending application has not gotten such an error does NOT mean that the data has actually been sent.
  - By default the send window is set to the peer's advertised receive window
    - Can use the `max_send_ws` tuning parameter to make it smaller

# TCP Stack – Send/Receive windows



- Application throughput is limited by
  - Receive window size / round trip time
    - $65535 / 0.075 \text{ s} = 873,800$
    - $65535 / 0.001 \text{ s} = 65,535,000$

# TCP Stack – Send/Receive windows



- Prior to 17.1
  - Maximum receive/send window size is 64K (65535 bytes)
  - System has 4 pools into which it assigns sockets
    - 64K, 32K, 16K and 8K
    - By default the number of sockets in any one pool is based on available memory. It can be adjusted up (or down) using STCP parameter
      - `set_stcp_param max_64k_windows NNN`
      - `set_stcp_param max_32k_windows NNN`
      - `set_stcp_param max_16k_windows NNN`
    - An application with multiple sockets can have sockets in different pools

# TCP Stack – Send/Receive windows



- The `list_stcp_params` will show you the maximum and the current numbers

```
list_stcp_params
. . . .
maximum send window size [4096-65535]    (max_send_ws)          65535 bytes
maximum 64k windows [>=0]                 (max_64k_windows)      10
current 64k windows                        1
maximum 32k windows [>=0]                 (max_32k_windows)      794
current 32k windows                        17
maximum 16k windows [>=0]                 (max_16k_windows)      0
current 16k windows                        0
```

# TCP Stack – Send/Receive windows



- Prior to VOS 16.0.2ag there was no way for an application to request placement in a pool
  - Applications are placed in these pools in a first come first serve basis
- Starting in VOS 16.0.2ag STCP supports the socket options `SO_RCVBUF` and `SO_SNDBUF`
  - Use of these options before 16.0.2ag would return the error number 6060
    - `e$streams_ENOPROTOOPT`. Protocol not available.

# TCP Stack – Send/Receive windows



- A socket will be placed in the 64K, 32K or 16K pool such that
  - The pool is the minimum of (16K, 32K, 64K) which is greater than the requested size
- and
- The number of sockets in pool has not reached the maximum value
  - If the indicated pool is full the next smaller pool is used
  - No error is returned if a smaller pool is used. The getsockopt function can be called to get the actual value

# TCP Stack – Send/Receive windows



- If the user running the application has write access to the `>system>acl>stcp_device` (or whatever ACL file is defined for the stcp device) the second condition is not checked.
  - This will allow the current to exceed the maximum value

# TCP Stack – Send/Receive windows



```
/          =name          stcp.m17
          =module_name    m17
          =device_type    streams
          =access_list_name stcp_access
          =streams_driver  stcp
          =clone_limit     5120
          =comment        Provides TCP API
```

```
display_access >system>acl>stcp_access -all
%azvos#m17_mas>system>acl>stcp_access
w  *.*
```

# TCP Stack – Send/Receive windows



- Starting in 17.1 STCP supports window scaling
  - This allows larger window to be used
  - Both sides of the connection must support window scaling
    - The window is calculated by taking the advertised window and shifting it the number of scale bits to the left so a window of 64K with a scale of 8 makes the actual window 16,776,960 bytes.
    - Window scale is determine when the connection is established
    - If only 1 side supports window scaling then it cannot be used and the largest window is 65535 bytes

# TCP Stack – Send/Receive windows



- Two new pools have been allocated
  - 256K
  - Huge
    - ◆ Anything over 256K

# TCP Stack – Send/Receive windows



```
maximum send window size [4096-1073725440] (max_send_ws)
1073725440
maximum huge windows [>=0] (max_huge_windows) 0
current huge windows 0
maximum 256k windows [>=0] (max_256k_windows) 25
current 256k windows 0
maximum 64k windows [>=0] (max_64k_windows) 203
current 64k windows 8
maximum 32k windows [>=0] (max_32k_windows) 813
current 32k windows 0
maximum 16k windows [>=0] (max_16k_windows) 0
current 16k windows
```

# TCP Stack – Send/Receive windows



- Space for buffers is not pre-allocated
- If many sockets have large windows and because of network problems those windows fill you can run out of streams memory
  - This is not a good thing
  - Defaults are based on memory configuration

# Application Design



# Application Design



- Application Turns
- Multiple writes
- Bypassing Nagle
- Window size

# Network Properties – Application Turns



- How many application messages must be transferred back and forth to get a user response
- Copy\_file across OSL is done in 4K chunks, sender sends 4K then waits for an OSL layer Ack. Transferring a 400K file requires 100 turns. At 75 ms a turn that is a minimum of 7500 ms, not counting any other overhead.

# Network Properties – Application Turns



```
ready; copy_file 400K_stream
%phx_cac#m2_user>Stratus>Noah_Davids>400k_stream
ready 13:56:57
ready 13:57:08
```

```
ftp 164.152.79.67
. . . .
409600 bytes sent in 2.02 seconds (198.24 Kbytes/sec)
```

- 11 seconds versus 2 seconds – turns hurt when latency is high

# Network Properties – Application Turns



- Application should be designed with a minimum number of turns
  - But life is full of tradeoffs
    - Just because a send completes is no guarantee that the data was actually sent, or received correctly, or read by the receiving application. The only way to confirm that is for the application to send an application layer ACK.



# <soap box mode>

- TCP is not message based, it is a stream of bytes
- TCP may combine small writes into 1 TCP segment or split large writes into multiple TCP segments
- A read with a 1000 byte buffer may return anything from 1 byte to 1000 bytes
- The application needs to parse the byte stream back into application messages

</soap box mode>

# Network Properties – Bypassing Nagle



- Since the application can't know what the MSS for a given connection is
- And the sending application has no control over the receiver's delayed ACK timer
- There is a way for the application to tell the TCP stack not to wait for small ( $<$  MSS) segments to be acknowledged

# Network Properties – Bypassing Nagle – multiple writes



- With Nagle

```
sc 6000 164.152.77.217 101 2
. . .
Transaction completed in 160 milliseconds
Transaction completed in 159 milliseconds
```

- Without Nagle

```
scnd 6000 164.152.77.217 101 2
. . .
Transaction completed in 72 milliseconds
Transaction completed in 72 milliseconds
```

# Network Properties – Bypassing Nagle – larger than 1 MSS



- With Nagle

```
sc 6000 164.152.77.217 600 1  
.  
.  
.  
Transaction completed in 158 milliseconds  
Transaction completed in 159 milliseconds
```

- Without Nagle

```
scnd 6000 164.152.77.217 600 1  
.  
.  
.  
Transaction completed in 74 milliseconds  
Transaction completed in 74 milliseconds
```

# Network Properties – Bypassing Nagle



- More tradeoffs
  - Doing this may increase protocol overhead on the network
    - Increased protocol overhead reduces bandwidth available to all applications

# Network Properties – Bypassing Nagle



- How to tell is Nagle is causing you problems
  - With packet\_monitor
  - Any network layer trace taken on the sender's network.

# Network Properties – Bypassing Nagle



```
scnd 6000 164.152.77.217 1000 1
```

```
. . . .
```

```
11:44:05.999 T 0536 TCP 134.111.200.229 164.152.77.217 60773 6000 A
11:44:06.094 R 0000 TCP 164.152.77.217 134.111.200.229 6000 60773 A

11:44:06.094 T 0464 TCP 134.111.200.229 164.152.77.217 60773 6000 PA
11:44:06.167 R 0011 TCP 164.152.77.217 134.111.200.229 6000 60773 PA

11:44:06.168 T 0536 TCP 134.111.200.229 164.152.77.217 60773 6000 A
11:44:06.250 R 0000 TCP 164.152.77.217 134.111.200.229 6000 60773 A

11:44:06.251 T 0464 TCP 134.111.200.229 164.152.77.217 60773 6000 PA
11:44:06.322 R 0011 TCP 164.152.77.217 134.111.200.229 6000 60773 PA

11:44:06.323 T 0536 TCP 134.111.200.229 164.152.77.217 60773 6000 A
11:44:06.406 R 0000 TCP 164.152.77.217 134.111.200.229 6000 60773 A

11:44:06.406 T 0464 TCP 134.111.200.229 164.152.77.217 60773 6000 PA
11:44:06.478 R 0011 TCP 164.152.77.217 134.111.200.229 6000 60773 PA
```

- Note that MSS sized packet followed by the naked (no data) ACK, followed by the rest of the data and then the application layer ACK.

# Network Properties – Bypassing Nagle



```
scnd 6000 164.152.77.217 1000 1
```

```
. . . .
```

```
11:46:49.230 T 0536 TCP 134.111.200.229 164.152.77.217 60775 6000 A
11:46:49.230 T 0464 TCP 134.111.200.229 164.152.77.217 60775 6000 PA
11:46:49.301 R 0000 TCP 164.152.77.217 134.111.200.229 6000 60775 A
11:46:49.302 R 0011 TCP 164.152.77.217 134.111.200.229 6000 60775 PA

11:46:49.303 T 0536 TCP 134.111.200.229 164.152.77.217 60775 6000 A
11:46:49.303 T 0464 TCP 134.111.200.229 164.152.77.217 60775 6000 PA
11:46:49.376 R 0000 TCP 164.152.77.217 134.111.200.229 6000 60775 A
11:46:49.376 R 0011 TCP 164.152.77.217 134.111.200.229 6000 60775 PA

11:46:49.377 T 0536 TCP 134.111.200.229 164.152.77.217 60775 6000 A
11:46:49.378 T 0464 TCP 134.111.200.229 164.152.77.217 60775 6000 PA
11:46:49.451 R 0000 TCP 164.152.77.217 134.111.200.229 6000 60775 A
11:46:49.451 R 0011 TCP 164.152.77.217 134.111.200.229 6000 60775 PA
```

- Note that MSS sized packet followed immediately by the rest of the data and then the remote's ACK and then application layer ACK.

# Network Properties – Bypassing Nagle



```
int nodelayFlag = 1;
. . . .
if (setsockopt (socks0, SOL_SOCKET, SO_NODELAY,
    (char *) &nodelayFlag, sizeof (nodelayFlag)) < 0)
{
    perror ("scnd: failed to set no delay");
    exit (errno);
}
```

- Note that the nodelay property flag is inherited, if it is set on a listening socket then any accepted sockets also have it set.

# Network Properties – Windows size



- Please refer to the slides regarding the Send/Receive Windows in the TCP Stack section for a discussion of release issues and how window size and round trip time affect application throughput.

# Network Properties – Windows size



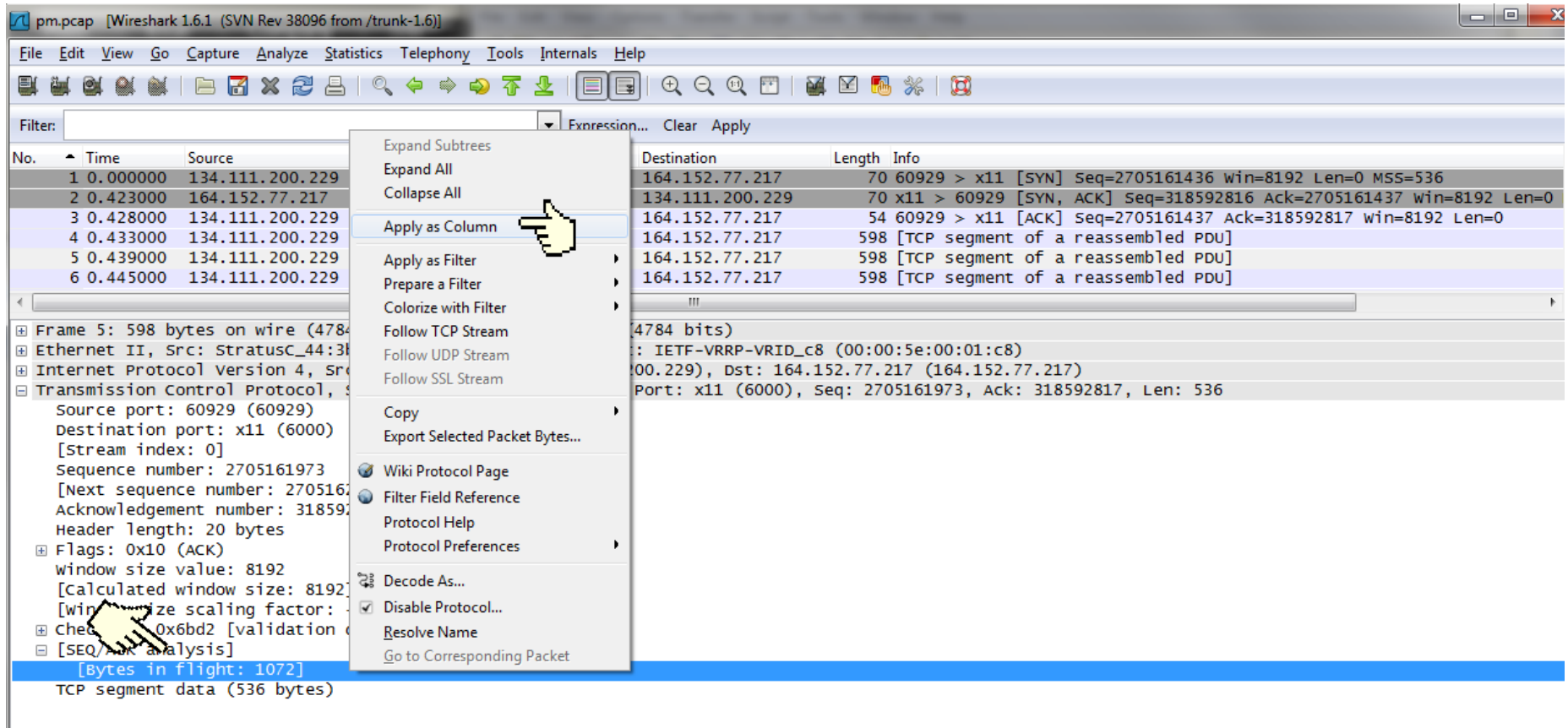
- Recognizing a window size issue is not that easy
  - It is NOT a zero window condition
    - Zero window is when the receiving application is not reading the data fast enough and the receiving TCP stack's receive buffer fills up
      - The receiving TCP stack advertises a zero window
  - You have to determine how much data is unacknowledged and compare that to the receive window

# Network Properties – Windows size



- You can take a packet\_monitor trace convert it to pcap format (see [http://www.noahdavids.org/self\\_publiched/pm2text2pcap.html](http://www.noahdavids.org/self_publiched/pm2text2pcap.html)) and then use wireshark to look at the bytes in flight
- You need to do this from the sending host

# Network Properties – Windows size



pm.pcap [Wireshark 1.6.1 (SVN Rev 38096 from /trunk-1.6)]

File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help

Filter: Expression... Clear Apply

No.	Time	Source	Destination	Length	Info
1	0.000000	134.111.200.229	164.152.77.217	70	60929 > x11 [SYN] Seq=2705161436 win=8192 Len=0 MSS=536
2	0.423000	164.152.77.217	134.111.200.229	70	x11 > 60929 [SYN, ACK] Seq=318592816 Ack=2705161437 win=8192 Len=0
3	0.428000	134.111.200.229	164.152.77.217	54	60929 > x11 [ACK] Seq=2705161437 Ack=318592817 win=8192 Len=0
4	0.433000	134.111.200.229	164.152.77.217	598	[TCP segment of a reassembled PDU]
5	0.439000	134.111.200.229	164.152.77.217	598	[TCP segment of a reassembled PDU]
6	0.445000	134.111.200.229	164.152.77.217	598	[TCP segment of a reassembled PDU]

Frame 5: 598 bytes on wire (4784 bits)  
Ethernet II, Src: StratusC\_44:38:00:00:00:00, Dst: 164.152.77.217 (164.152.77.217)  
Internet Protocol Version 4, Src: 134.111.200.229, Dst: 164.152.77.217 (164.152.77.217)  
Transmission Control Protocol, Src Port: 60929 (60929), Dst Port: x11 (6000), Seq: 2705161973, Len: 536

- Expand Subtrees
- Expand All
- Collapse All
- Apply as Column
- Apply as Filter
- Prepare a Filter
- Colorize with Filter
- Follow TCP Stream
- Follow UDP Stream
- Follow SSL Stream
- Copy
- Export Selected Packet Bytes...
- Wiki Protocol Page
- Filter Field Reference
- Protocol Help
- Protocol Preferences
- Decode As...
- Disable Protocol...
- Resolve Name
- Go to Corresponding Packet

[Bytes in flight: 1072]  
TCP segment data (536 bytes)

# Network Properties – Windows size



pm.pcap [Wireshark 1.6.1 (SVN Rev 38096 from /trunk-1.6)]

File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help

Filter: Expression... Clear Apply

No.	Time	Source	Destination	Protocol	Destination	Length	Bytes in flight	Info
10	0.858000	134.111.200.229	164.152.77.217	TCP	164.152.77.217	598	536	[TCP segment of a reassembled PDU]
11	0.863000	134.111.200.229	164.152.77.217	TCP	164.152.77.217	598	1072	[TCP segment of a reassembled PDU]
12	0.868000	134.111.200.229	164.152.77.217	TCP	164.152.77.217	598	1608	[TCP segment of a reassembled PDU]
13	0.873000	134.111.200.229	164.152.77.217	TCP	164.152.77.217	598	2144	[TCP segment of a reassembled PDU]
14	0.878000	134.111.200.229	164.152.77.217	TCP	164.152.77.217	598	2680	[TCP segment of a reassembled PDU]
15	0.897000	134.111.200.229	164.152.77.217	TCP	164.152.77.217	598	3216	[TCP segment of a reassembled PDU]

Frame 5: 598 bytes on wire (4784 bits), 598 bytes captured (4784 bits)

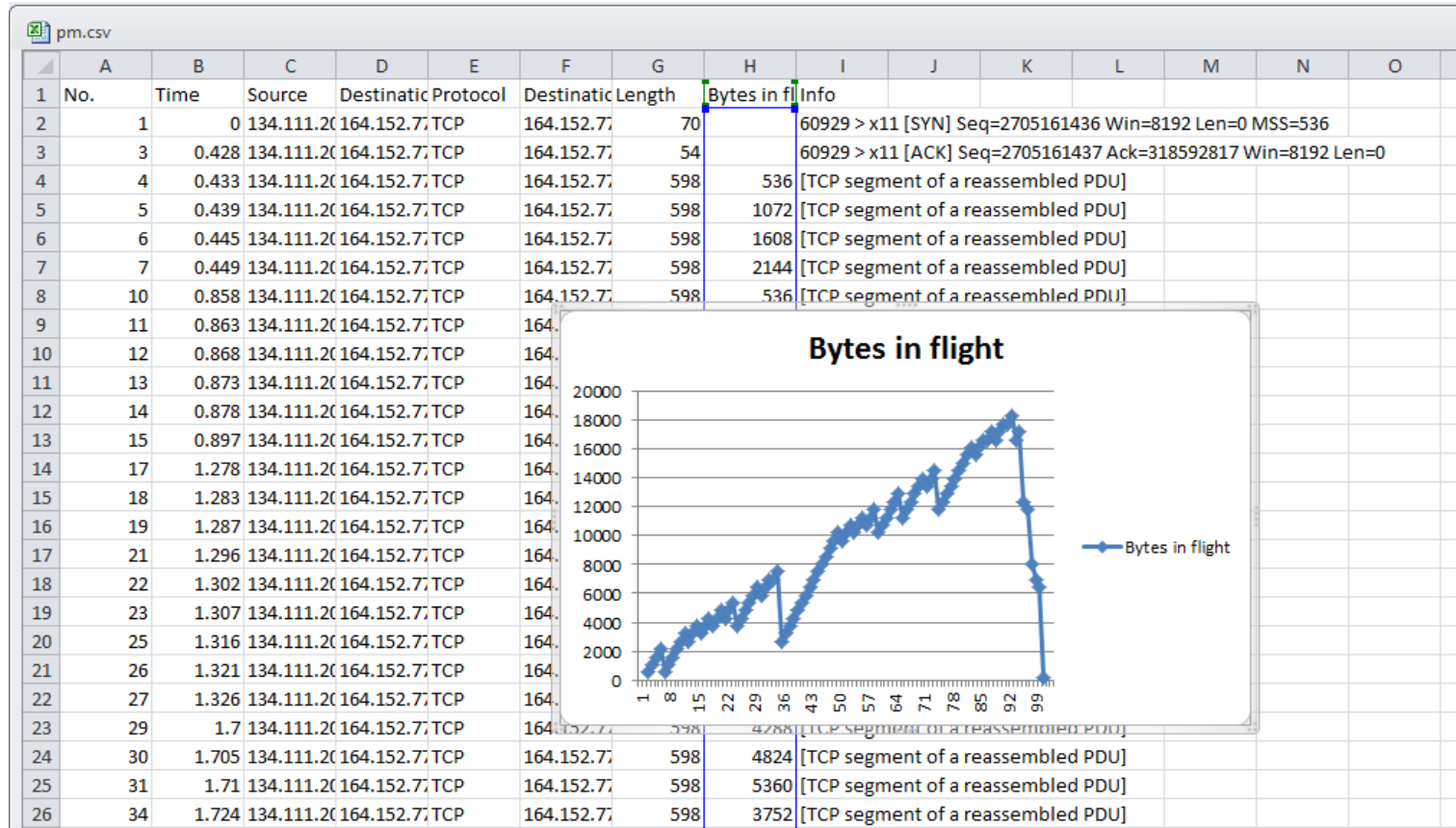
- Ethernet II, Src: StratusC\_44:3b:6c (00:00:a8:44:3b:6c), Dst: IETF-VRRP-VRID\_c8 (00:00:5e:00:01:c8)
- Internet Protocol Version 4, Src: 134.111.200.229 (134.111.200.229), Dst: 164.152.77.217 (164.152.77.217)
- Transmission Control Protocol, Src Port: 60929 (60929), Dst Port: x11 (6000), Seq: 2705161973, Ack: 318592817, Len: 536
  - Source port: 60929 (60929)
  - Destination port: x11 (6000)
  - [Stream index: 0]
  - Sequence number: 2705161973
  - [Next sequence number: 2705162509]
  - Acknowledgement number: 318592817
  - Header length: 20 bytes
  - Flags: 0x10 (ACK)
  - window size value: 8192
  - [Calculated window size: 8192]
  - [window size scaling factor: -2 (no window scaling used)]
  - Checksum: 0x6bd2 [validation disabled]
  - [SEQ/ACK analysis]
  - [Bytes in flight: 1072]
  - TCP segment data (536 bytes)

# Network Properties – Windows size



- From here you can either eyeball the values or
- Export the file as a CSV file, read it into your favorite spreadsheet program and graph the Bytes in flight column

# Network Properties – Windows size



- This doesn't show a window problem but I couldn't create an example that did

# Network Properties – Windows size



- Applications with multiple turns, like OSL will not have a window size problem unless the window is less than the amount of data in a turn
  - For example with OSL that would be 4K.

# Network Properties – Windows size



```
int recv_buf = 131070;
. . . .
if (setsockopt (socks[1], SOL_SOCKET, SO_RCVBUF,
    (char *) &recv_buf, sizeof (recv_buf)) < 0)
{
    perror ("echo_server: Error setting receive buffer size");
    exit (errno);
}
```

- Note that the buffer size property is NOT inherited, accepted sockets have the default size and any application specific size must be explicitly set.

# Network Emulation Tools



# Network Emulation Tools



- WANem
- Dummynet

# Network Emulation Tools - WANem



- <http://wanem.sourceforge.net/>
- Download an ISO image and boot it
  - GUI interface reached via a web browser
  - May need to drop into command line to set up routes
- Can be used on a PC with just 1 interface
  - 2 interfaces is obviously better

# Network Emulation Tools - WANem



- Can specify and source and/or destination IP network or address and/or any port
  - Cannot specify a particular protocol
- Can specify delay, loss, duplication, reorder, corruption, jitter

# Network Emulation Tools - WANem



**WANem v2.3 is released with updated Copyright**

WANem is a Wide Area Network Emulator, meant to provide a real experience of a Wide Area Network/Internet, during application development / testing over a LAN environment. Typically application developers develop applications on a LAN while the intended purpose for the same could be, clients accessing the same over the WAN or even the Internet. WANem thus allows the application development team to setup a transparent application gateway which can be used to simulate WAN characteristics like Network delay, Packet loss, Packet corruption, Disconnections, Packet re-ordering, Jitter, etc. WANem can be used to simulate Wide Area Network conditions for Data/Voice traffic and is released under the widely acceptable GPL v2 license. WANem thus provides emulation of Wide Area Network characteristics and thus allows data/voice applications to be tested in a realistic WAN environment before they are moved into production at an affordable cost. WANem is built on top of other FLOSS[Free Libre and OpenSource] components and like other intelligent FLOSS projects has chosen not to re-invent the wheel as much as possible.

From a functionality perspective WANem hooks into the Linux kernel towards provisioning the network emulation characteristics and extends the functionality with additional modules. Based on a re-mastered Knoppix cd WANem allows quick and easy setup in any development environment with an intuitive web interface for purposes of configuration.

WANEM is [Open Source](#) and licensed under the [GNU General Public License](#). You are free to download WANem and use it in your own environments. We encourage you to write to us using the SourceForge forums. You may let us know your perspective on scope for improvement, or if you would like to contribute in anyway possible or ofcourse just to drop us a note of encouragement.

Developed by: [Manoj Nambiar](#), [Hemanta Kumar Kalita](#), [Debadatta Mishra](#) and [Shirish Rane](#) at Performance Engineering Research Centre, [TATA Consultancy Services](#), Mumbai.

# Network Emulation Tools - WANem



Firefox | Useful Scripts | Gmail - Local contact for Sonya Davi... | WANEM : The Wide Area Network E... | WANEM : The Wide Area Network E... | 164.152.77.107/WANem/ | wanem

**TATA** **WANem** [Home](#)  
**TATA CONSULTANCY SERVICES** *The Wide Area Network Emulator*  
 Performance Engineering Research Centre

[About](#) [WANalyzer](#) [Basic Mode](#) [Advanced Mode](#) [Save/Restore](#) [Help](#)

WANem is running

<b>Interface: eth0</b>		<b>Packet Limit</b> 1000 (Default=1000)				<b>Symmetrical Network:</b> Yes ▾	
<b>Bandwidth</b>	Choose BW	Other ▾				Other: Specify BW(Kbps) 0	
<b>Delay</b>		<b>Loss</b>		<b>Duplication</b>		<b>Packet reordering</b>	
Delay time(ms)	150	Loss(%)	0	Duplication(%)	0	Reordering(%)	0
Jitter(ms)	0	Correlation(%)	0	Correlation(%)	0	Correlation(%)	0
Correlation(%)	0					Gap(packets)	0
Distribution	-N/A-						
<b>Idle timer Disconnect</b>	Type	none ▾	Idle Timer			Disconnect Timer	
<b>Random Disconnect</b>	Type	none ▾	MTTF Low		MTTF High	MTTR Low	MTTR High
<b>Random connection Disconnect</b>	Type	none ▾	MTTF Low		MTTF High	MTTR Low	MTTR High
IP source address	any	IP source subnet		IP dest address	any	IP dest subnet	Application port if any
							any

Display commands only, do not execute them

# Network Emulation Tools - Dummynet



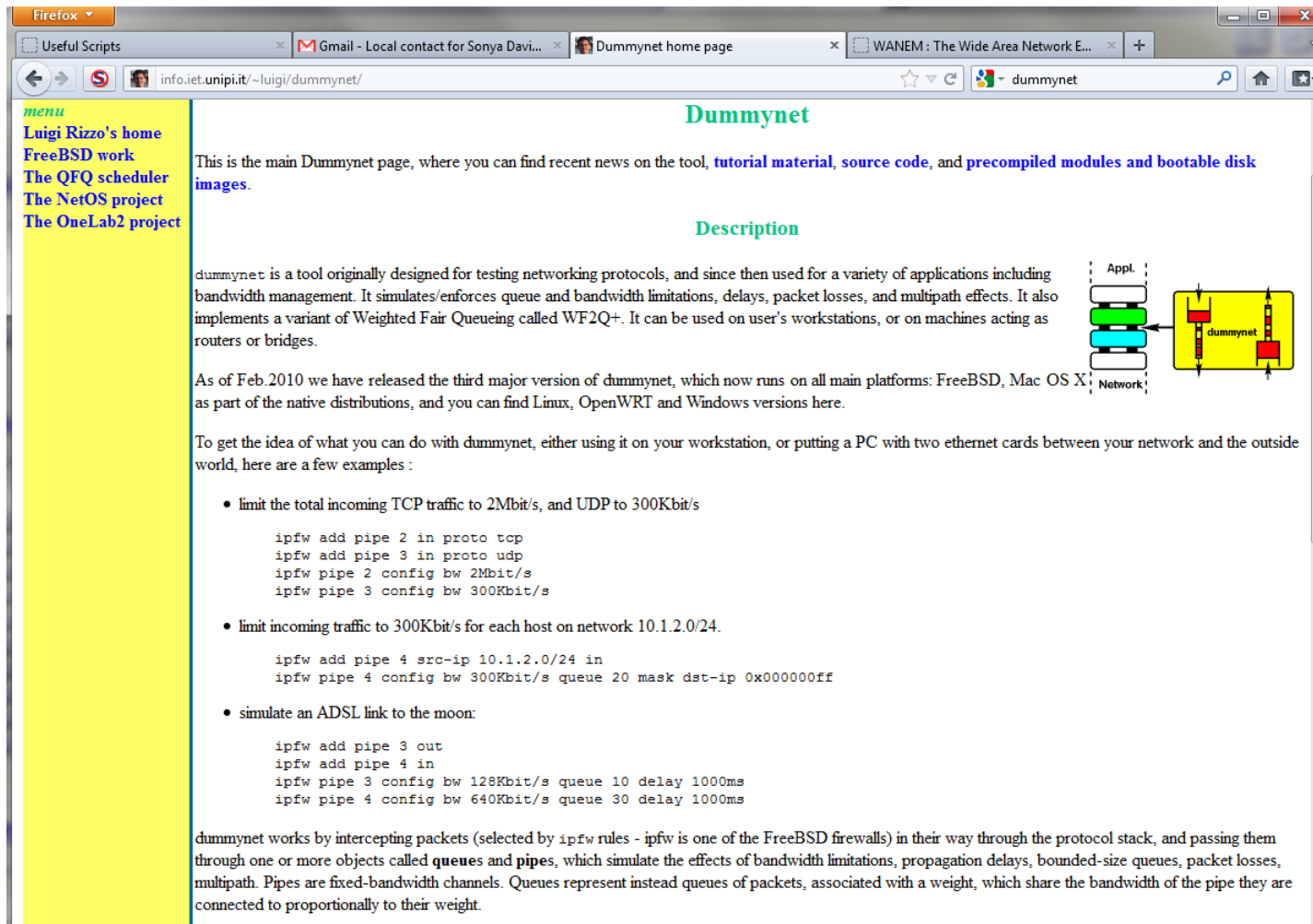
- <http://info.iet.unipi.it/~luigi/dummynet/>
- Can run on multiple Osss
  - FreeBSD, Mac OS, Linux, Windows
- Download an ISO image and boot it
  - Based on PicoBSD
  - Command line interface
- Can be used on a PC with just 1 interface
  - 2 interfaces is obviously better

# Network Emulation Tools - Dummynet



- Very flexible in specifying source and destination and protocols

# Network Emulation Tools - Dummysnet



**menu**  
Luigi Rizzo's home  
FreeBSD work  
The QFQ scheduler  
The NetOS project  
The OneLab2 project

## Dummysnet

This is the main Dummysnet page, where you can find recent news on the tool, [tutorial material](#), [source code](#), and [precompiled modules and bootable disk images](#).

### Description

dummysnet is a tool originally designed for testing networking protocols, and since then used for a variety of applications including bandwidth management. It simulates/enforces queue and bandwidth limitations, delays, packet losses, and multipath effects. It also implements a variant of Weighted Fair Queueing called WF2Q+. It can be used on user's workstations, or on machines acting as routers or bridges.

As of Feb.2010 we have released the third major version of dummysnet, which now runs on all main platforms: FreeBSD, Mac OS X as part of the native distributions, and you can find Linux, OpenWRT and Windows versions [here](#).

To get the idea of what you can do with dummysnet, either using it on your workstation, or putting a PC with two ethernet cards between your network and the outside world, here are a few examples :

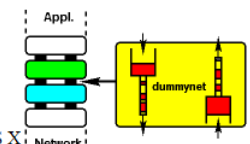
- limit the total incoming TCP traffic to 2Mbit/s, and UDP to 300Kbit/s

```
ipfw add pipe 2 in proto tcp
ipfw add pipe 3 in proto udp
ipfw pipe 2 config bw 2Mbit/s
ipfw pipe 3 config bw 300Kbit/s
```
- limit incoming traffic to 300Kbit/s for each host on network 10.1.2.0/24.

```
ipfw add pipe 4 src-ip 10.1.2.0/24 in
ipfw pipe 4 config bw 300Kbit/s queue 20 mask dst-ip 0x000000ff
```
- simulate an ADSL link to the moon:

```
ipfw add pipe 3 out
ipfw add pipe 4 in
ipfw pipe 3 config bw 128Kbit/s queue 10 delay 1000ms
ipfw pipe 4 config bw 640Kbit/s queue 30 delay 1000ms
```

dummysnet works by intercepting packets (selected by `ipfw` rules - `ipfw` is one of the FreeBSD firewalls) in their way through the protocol stack, and passing them through one or more objects called **queues** and **pipes**, which simulate the effects of bandwidth limitations, propagation delays, bounded-size queues, packet losses, multipath. Pipes are fixed-bandwidth channels. Queues represent instead queues of packets, associated with a weight, which share the bandwidth of the pipe they are connected to proportionally to their weight.



# Summary



# Network Application Performance - Summary



- Network Properties
  - Latency
    - Know what your worse case will be and plan for it
  - Loss/Corruption
    - Measure/monitor and be prepared to complain if it gets too high ( $> 1\%$ )
  - Bandwidth
    - Know what the application will use and what your total capacity is
    - Loss/Corruption monitoring will tell you if you exceed your bandwidth

# Network Application Performance - Summary



- TCP Stack
  - Maximum Segment Size (MSS)
    - Understand when STCP will use 536 bytes instead of 1460
    - Tune MSS upwards if your network can handle it
  - ACK delay
    - Under certain circumstances you might want to think about using a smaller ACK delay but only as a last resort
  - Jumbo Frames
    - Might buy you some performance improvement for local applications
  - Send/Receive windows
    - Understand how they are set and throughput limits they impose
    - Consider upgrading so application can be in control

# Network Application Performance - Summary



- Application Design
  - Application Turns
    - Reduce turns as much as possible
  - Multiple writes
    - Reduce writes as much as possible
  - Bypassing Nagle
    - Consider setting `SO_NODELAY`
  - Window size
    - Understand the throughput limits they impose
    - Consider upgrading so application can be in control

# Questions



# Network Application Performance



- A copy of this power point presentation can be found at [http://www.noahdavids.org/webinars/network\\_application\\_performance\\_2012-01-18.pdf](http://www.noahdavids.org/webinars/network_application_performance_2012-01-18.pdf)
- A recording of this presentation can be found on the Stratus channel at YouTube - StratusTech

**Thank You!**

