Enabling High-Performance Internet-Wide Measurements on Windows

Matt Smith

Joint work with Dmitri Loguinov

Internet Research Lab
Department of Computer Science and Engineering
Texas A&M University

April 9, 2010

Agenda

- Introduction
  - Background and Motivations
- Windows / Linux Network Stacks: An Overview
- Our Approach: IRLstack
- Performance Evaluation
- Conclusion

Introduction

- As the Internet continues to grow, capturing accurate large-scale measurements remains an important research problem
  - How big is the web? Can we capture a “snapshot” of a P2P network? Etc.
- Distributed server clusters are often used in commercial applications
  - May not be available to academic researchers
- Bottlenecks in measurement projects are often encountered at the *client*-side
  - Rate at which requests can be issued

Motivations

- A few of our representative projects which require high sustained rates of measurement traffic:
  - P2P network analysis (Gnutella crawler)
  - IRLbot web crawler
  - DNS infrastructure traversal
  - Service discovery using horizontal scanning
- All these projects measure networks which constantly evolve during the measurement period
- Other applications (e.g., IDS, monitoring tools) also benefit from a scalable network stack

Motivations

- Our goal: wire-rate transmission and capture of packets of all sizes
- Similar work has been done on Linux platforms; however no serious efforts have used Windows thus far
- We show that Windows can be used as a platform for serious high-performance research as well

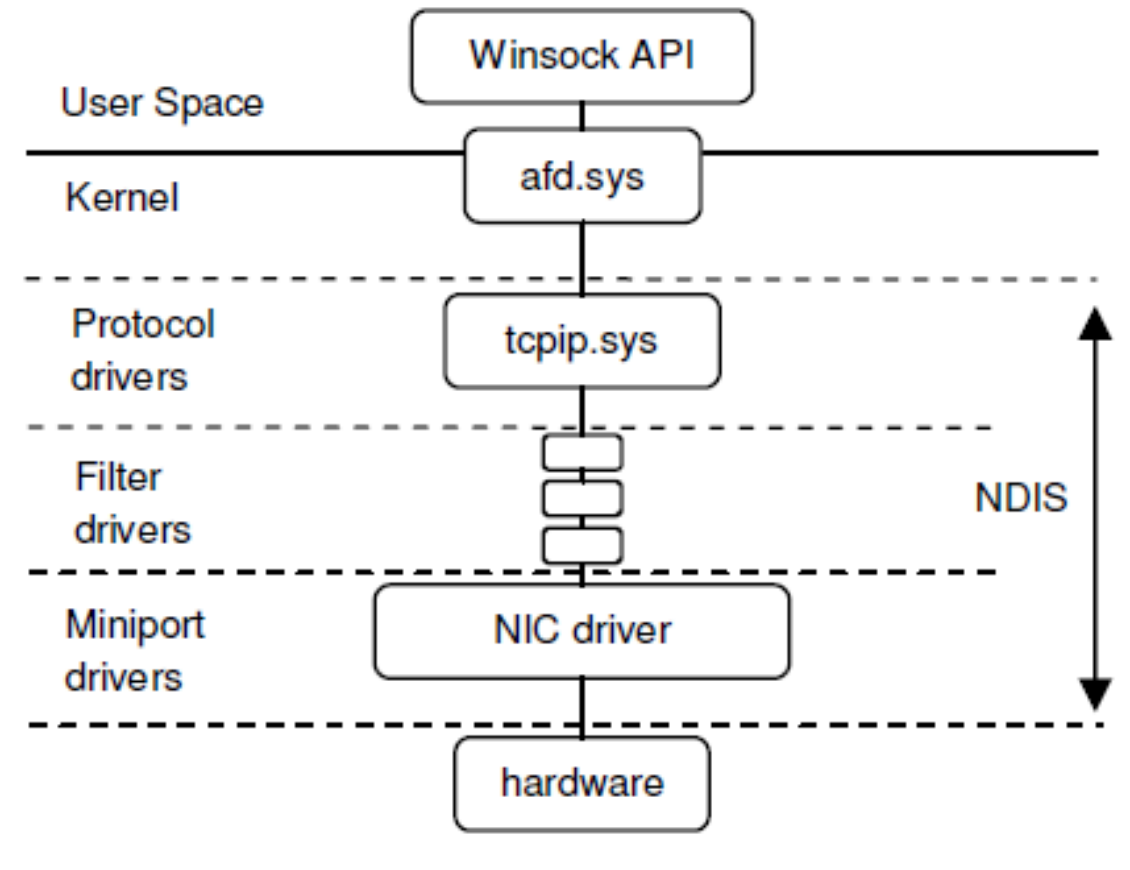
Agenda

- Introduction
  - Background and Motivations
- Windows / Linux Network Stacks: An Overview
- Our Approach: IRLstack
- Performance Evaluation
- Conclusion

Windows Network Stack Overview

- Three classes of drivers which implement different layers of functionality
  - NDIS: Network Driver Interface Specification
- Protocol drivers: accept requests from user-space, construct link-layer frames as appropriate
- Filter drivers: receive and possibly process any frames sent to or from the NIC
  - Note that WinPcap is implemented as a filter driver with a direct interface to user-space
- Miniport drivers: specific to each NIC, directly interface with hardware; DMA, interrupts, etc.

Windows Network Stack Overview



Computer Science, Texas A&M University

Linux Network Stack Overview

- Our primary Linux comparison focuses on the modified stack available from the *ntop* project, which makes two main contributions
- PF_RING: uses DMA and Intel I/OAT for zero-copy availability of kernel memory buffers in user-space
- TNAPI: deserializes receive operations utilizing multiple RX queues and making them available concurrently

Performance Evaluation: Winsock and WinPcap

- As mentioned earlier, several of our projects would benefit from very high send/receive rates
  - Hundreds of thousands of packets per second (pps); implies small packets and more overhead
  - Ideally wire rate: 1,488,095 pps for Gigabit Ethernet
- Maximum single-NIC transmit rates on our test system left much to be desired, despite 100% CPU usage
  - Winsock best case (raw IP socket, single destination): 207 kpps
  - WinPcap: 50 kpps

Performance Evaluation: Winsock and WinPcap

- What bottlenecks prevent Winsock/WinPcap from achieving wire-rate performance?
  - For Winsock, primarily IP table lookups
  - At a common lower level (NDIS), synchronization spinlock overhead required for every transfer
  - The amount of “overhead” work per packet (irrespective of length) is very high
- With these constraints we cannot achieve wire rate with minimum-sized packets – which measurement traffic tends to be

Agenda

- Introduction
  - Background and Motivations
- Windows / Linux Network Stacks: An Overview
- Our Approach: IRLstack
- Performance Evaluation
- Conclusion

Our Approach: IRLstack

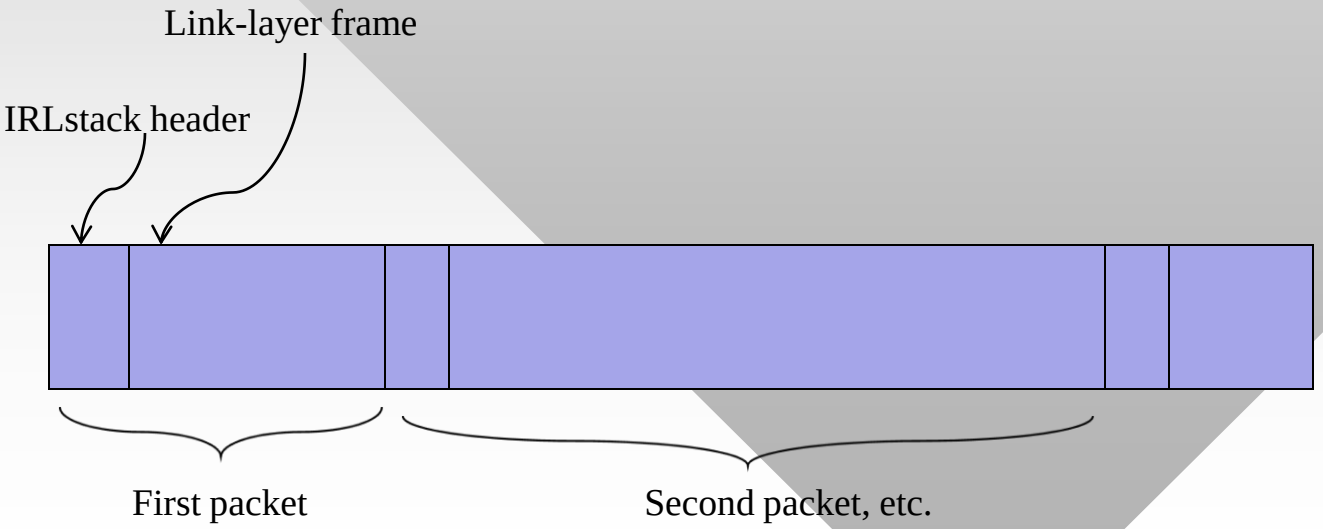
- Goal: build a very high-performance interface for sending/receiving packets directly from a NIC
  - Applications can use this interface directly (i.e., a "raw socket" interface)
  - Alternately, an intermediate layer (e.g., simplified TCP stack) can sit between IRLstack and the user-space application
- Any per-packet processing that isn't absolutely necessary is not included
- Batching of many packets (sending or receiving) maximizes useful work per request

Our Approach: IRLstack

- IRLstack is implemented as an NDIS driver stack with direct access from user space
  - E.g., ReadFile/WriteFile APIs
- Two components of the stack
  - IRLstackP.sys *protocol driver* – user applications open a handle to this driver; processes send/receive requests
  - IRLstackF.sys *filter driver* – intercepts all incoming packets, redirects to IRLstackP.sys as appropriate
  - The filter driver uses a list of IP addresses assigned to IRLstack to rewrite link-layer frame headers and perform the redirection

Our Approach: IRLstack

- To accommodate packet batching, IRLstack request buffers consist of a series of complete link-layer (usually Ethernet) frames
  - A small header specific to IRLstack precedes each frame
  - The same format is used on both TX and RX paths



Our Approach: IRLstack

- Other design notes:
  - Multiple requests can be outstanding at a time
  - Checksums usually need not be calculated in software (if using a NIC with checksum offloading)
  - An entire buffer (hundreds or thousands of packets) is processed as a single request at all levels in the kernel
  - Zero-copy send path
  - IRLstack coexists with the Windows network stack on a single adapter; default configuration requires an extra IP address to be assigned to the interface

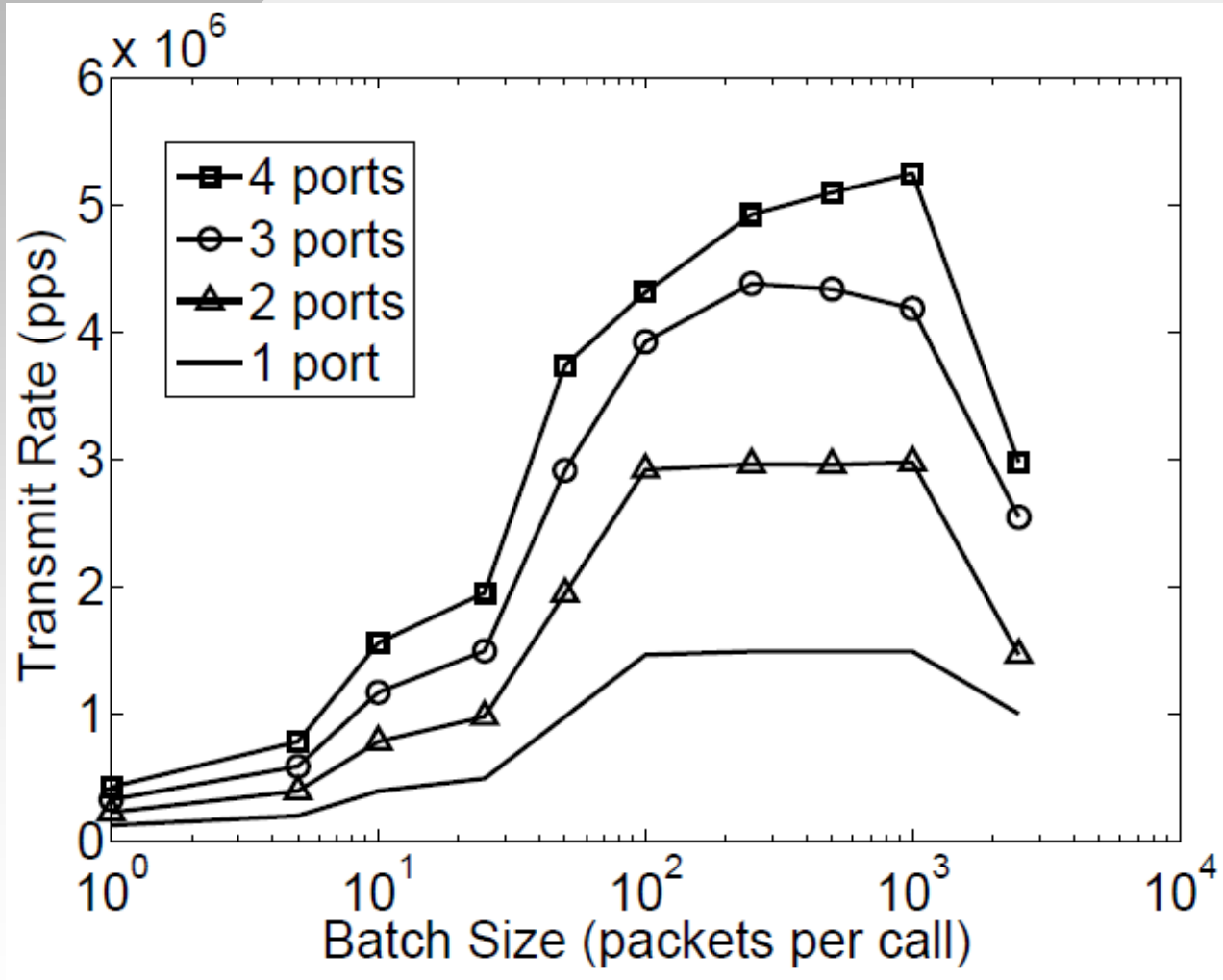
Agenda

- Introduction
  - Background and Motivations
- Windows / Linux Network Stacks: An Overview
- Our Approach: IRLstack
- Performance Evaluation
- Conclusion

Performance Evaluation

- Our testing focuses on minimum-size packets
  - Reflects the properties of much of our measurement traffic
  - This is the “hardest” scenario – most overhead
- Optimal transmission rate occurs around 512 packets per call on our test setup (Intel Pro/1000 PT network adapter)
  - Batch sizes below 100 packets are unable to fully utilize a gigabit link; less work is done per trip down the network stack
  - Large batches (thousands of packets) actually start to lose performance, which we attribute to the miniport

Performance Evaluation



Performance Evaluation

- Receive performance is somewhat (~20-50%) slower than send performance
  - Not zero-copy at the moment
  - Interrupt frequency is higher and batch size is lower (e.g., 64); this is out of our direct control as miniport drivers on Windows are typically not open-source
- For reference we look to the Linux numbers from the *ntop* project; IRLstack's performance compares favorably on similar hardware

Method	Rate (pps)			
	1 port / 1 core	2 ports / 2 cores	3 ports / 3 cores	4 ports / 4 cores
IRLstack	1,232,745 (82%)	1,526,460 (51%)	2,282,554 (51%)	2,946,707 (50%)
Linux [5]	~ 920,000 (61%)	–	–	~ 3,000,000

Agenda

- Introduction
  - Background and Motivations
- Windows / Linux Network Stacks: An Overview
- Our Approach: IRLstack
- Performance Evaluation
- Conclusion

Conclusion

- More information about latency and TCP performance can be found in the full paper
- Windows has traditionally been avoided as a research platform; however with a well-designed driver suite such as IRLstack this need not be the case
- Areas of future work:
  - Further optimization on receive path
  - Evaluation on 10 Gigabit Ethernet NICs and with other new hardware features (e.g., DMA remapping)