Stress-Testing a Gbps Intrusion Prevention Device on DETER

Nicholas Weaver
Vern Paxson
ICSI

Acknowledgements

- Joint work with Jose “Chema” Gonzalez
- Sponsored by
  - NSF/DHS ANI-0335290 (EMIST)
  - DOE DE-F602-04ER25638 (Shunting)
  - NSF CNS-0433702 (CCIED)
  - IP/tool support from Xilinx University Program
- Opinions are of the authors, not of the funders

The problem: Testing a Gbps IPS device

- We have a a prototype hardware design for an IPS device, the ***Shunt***
  - Interacts with a software IDS (Bro)
  - Designed to be an inline device on Gigabit networks
  - Built on the NetFPGA 2.0 development platform
- We can perform functional testing on our desktop
- But we need performance/stress testing
  - Need to ensure that the hardware can actually operate at Gigabit line rate
  - Both to discover bugs and confirm limitations

Outline

- What is the NetFPGA?
  - Current NetFPGA Integration into DETER
  - Final NetFPGA Integration Plans
- What is the Shunt?
- Stress Testing the Shunt
  - Performance testing with **`iperf`**
  - Out-of-order packets
  - Worst Case Cache Behavior
  - Expected (and enhanced expected) Case Behavior

The NetFPGA 2.0 Board

- Developed at Stanford by Greg Watson, Nick McKeown and Martin Casado
  - Primarily as a teaching platform
  - But enough performance to be an interesting research platform
- Xilinx Virtex 2 Pro 30 FPGA coupled to a 4x Gigabit Ethernet PHY, 2x 2MB SRAMs, and a PCI controller
  - Existing design (***CNET***) is a 4 port GigE Ethernet NIC
- 2 Gbps, 32b wide, 62.5MHz , point-to-point internal datapaths
  - Relatively easy to meet timing
  - 2x bandwidth makes it relatively easy to meet network bandwidth requirements (theoretically)

Why NetFPGA?

- A large enough FPGA to be "interesting"
  - Can implement 4 Gigabit Ethernet MACs and control logic with enough room for further logic
- A small enough FPGA to be "reasonable"
  - Only 30,000 logic cells and 2 Mb of on-chip memory
  - ~$600 each in single-unit quantity (-5 speed grade) from Digikey
- Enough memory to be "interesting"
  - 2 MB of usable SRAM
    - Other 2 MB are reserved for the host interface
- Small enough memory to be "reasonable"
  - Memory cost ~$60 in single-unit quantity
- Tight coupling to the host
  - Board appears as a quad-port Ethernet
  - Peak/poke interface to the SRAM

Current NetFPGA Integration In DETER

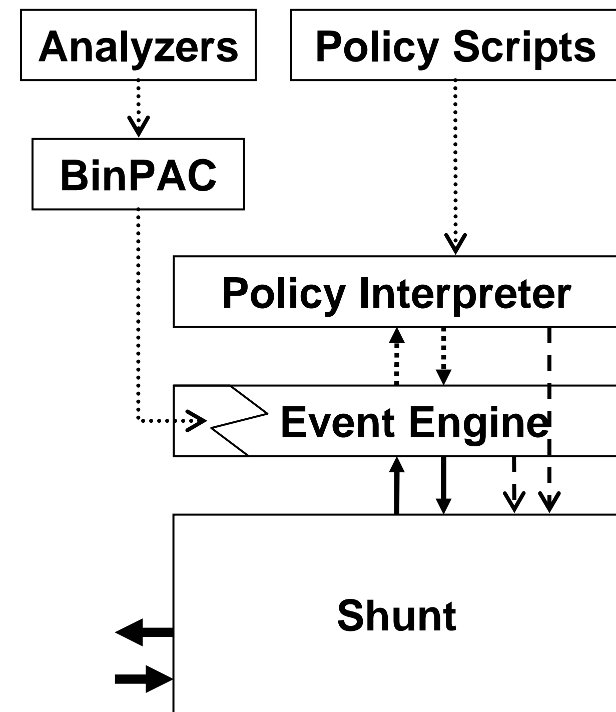
- A standalone system
  - Running a fully updated Fedora Core 3
  - NetFPGA driver software
  - ***Not*** managed by DETER
- Why standalone?
  - Currently the only user
    - So group management not necessary yet
  - Driver is currently somewhat finicky
    - Wanted the system working now for these experiments
- Two NetFPGA ports managed as separate devices
  - Emulab recognises and can assign these ports into the network as part of an experiment
    - Each port needs to be a standalone "device", as ports on the NetFPGA may ***not*** be symmetric depending on the user's design

Planned NetFPGA Integration In DETER

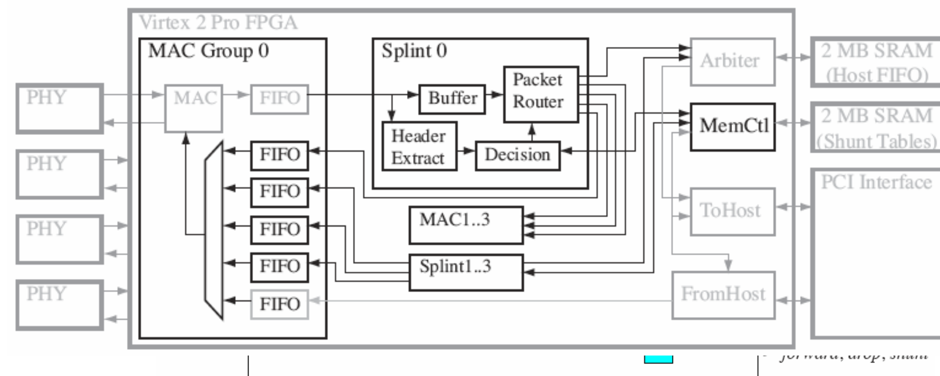
- The Fedora Core image with Emulab-modified kernel will build and properly install the NetFPGA driver
  - A good sign
  - Thus will create a modified image for general use
- Compilation environment will currently be unsupported
  - Licensing issues with making the Xilinx tools generally available
  - However, academics can generally get licenses for the necessary tools and IP from the Xilinx University Program
- Version 2.1 should be available in December
  - Hope to widely deploy the revised board in DETER and elsewhere

Shunting

- A new network interface and filter for Bro: the Shunt
  - The Shunt is an ***inline*** element, all packets pass through it
    - Packets are reinjected by the IDS after they are analyzed Allows attacks to be blocked in progress
- The Shunt's filtering is programmable
  - For each {host, connection, packet header}:
    - Forward the packet onward (whitelist)
    - Drop the packet (blacklist)
    - Sample the packet
      - Allow nonintrusive monitoring
    - Shunt the packet to the host
  - Each match has a priority:
    - Select highest priority match
  - The IDS can add and remove entries from the table

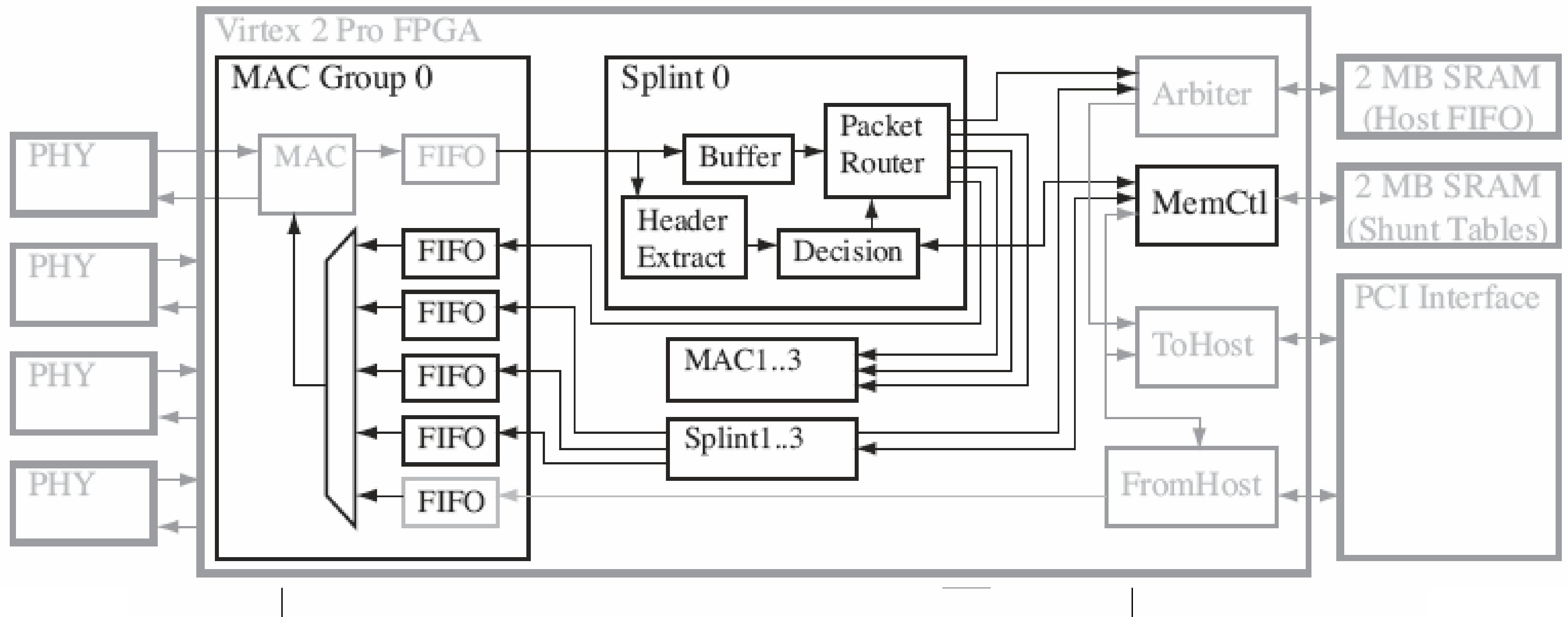


Shunting Operation



- Extended Bro to use the Shunt
  - New API calls and default behavior for connections
- A hardware element on NetFPGA
  - Look up packet in every table
    - Select highest priority math
    - If no match, default is to shunt
  - Very easy for hardware:
    - Single memory location per table
    - Priority encoder to select action
- A software element (the shim) to manage the shunt
  - Assumes that the hardware shunt may have errors
  - Manages the hardware cache
- Bro policies need to be shunt-aware
  - Tell Shunt to forward the connection when no further analysis is possible

The Figures in More Detail



Testing the Shunt: *Click* test harness

- Rather than the full Bro IDS, we are testing with a small harness written in ***Click***
  – Any non-IP packet is simply passed between the two ports
  – Any IP packet has the connection table entry set and is then passed
- Thus the first packet of the connection stresses the low level software path
  – All subsequent packets go through the hardware path

Test 1: Throughput

- We know the software path is low bandwidth
  - A single PCI 33 MHz/32b interface
- But the hardware path ***should*** be full gigabit
- We used the ***iperf*** bandwidth testing tool
  - Sends either a maximum-rate TCP stream or a programmable rate UDP stream to a lightweight server
  - We use the UDP mode
    - It allows us to send at a given rate

Results: Bugs discovered

- Currently, <450 Mbps can be easily sent and received
  - Passes through the board without a problem
- But the input FIFO locks up at >450 Mbps
  - ***Not*** a known bug, but an unknown bug
- Actually 3+ bugs (currently working on)
  - Input FIFO locks up under high load conditions
  - Input FIFO sometimes drops the first byte under high load conditions
  - Design doesn't appear to be processing at full rate
    - FIFO appears to be overflowing even when it shouldn't
  - Output FIFO locks up on Ethernet MTU-sized packet (sometimes)

Stress-testing is essential

- We can test correctness in a lower volume environment
  - Can send packets at a low rate and ensure that the proper responses are seen
- But stress-testing requires substantial resources:
  - I don’t *have* a good Gigabit source/sink on my desktop, let alone 6 in the current configuration
  - Using 3 GHz, dual processor, ***good bus*** systems
    - You can’t really source Gigabit traffic on a commodity PC with standard PCI, it requires PCI-X or PCI-E based servers

Test 2:
Out of Order packets

- TCP reacts poorly to significantly out-of-order packets
  - Treats as a packet loss
  - Some UDP protocols may also react badly depending on the implementation
- The question: if a high-volume TCP stream is evicted from the connection cache, will this cause a problem?
  - The first few packets go through the software path, before the rule can be reinserted into the connection cache
- ***Iperf*** reports out-of-order packets
  - Send at maximum rate for UDP
    - Since UDP test starts at full rate, this ends up being equivelent to a cache miss on a high volume TCP flow
  - See # of reported packets out-of-order

Out of Order results

- At ~450 Mbps, MTU sized packets
  - The 13th packet is processed in hardware, arrives before the first 12 packets
- QED: Evicting a TCP stream which uses large packet trains would be a problem!
  - But false evicts of lower-volume streams are not a significant problem
    - Testing with MTU-sized packets is OK
- Cache management will be important in deployment
  - Must sample to ensure that high-volume streams are not evicted
    - Will not work if a TCP stream alternates between very high volume and low volume
      - But this ***should*** not happen
        and if it does, its probably ok if ***that stream*** sees out of order packets
    - Relatively easy to implement
  - Also suggests slightly different cache organization

Test 3: Stressing the Cache

- Not yet started, but a simple idea
  - Use a group of senders and receivers
  - For each packet, select {sequentially | random} src_port, dst_port, dst_system
    - Send UDP packet to destination
  - Each tuple represents an independent connection
- This behavior will stress the cache to the limit
  - Mostly looking for lock-up bugs comparable to what we saw on high-volume traffic
    - We know the throughput will suffer, but packet drops are OK in such conditions

Test 4: Expected-Case Stress

- What happens if normal traffic is just increased in volume and/or variability?
  - How does the cache behave?
- We have already developed TCP-based source models
  - Based on traces of LBNL's enterprise network
  - Used to test AC-TRW worm-defense written in Click
- Use these to observe behavior under expected-case stress
  - Increase the number of connections to increase the load
- Test not yet performed
  - Will require a large number of DETER systems
    - Need physical nodes to produce proper network traffic
    - Need a 1-1 representation for this to work properly on all caches
      - Address caches as well as connection caches

Conclusions

- Stress testing is a very important test
- It is actually harder than it sounds:
  - You need sufficient resources
    - Gigabit traffic generation needs high-end systems
  - You need to understand the target being tested
    - In our case: what is the cache behavior
    - There is no silver bullet here
  - You need to instrument the system to know you are generating valid testing traffic
- Design the tests to target the device
  - Stress-testing becomes target-specific tests: not everything can be reused between different experiments