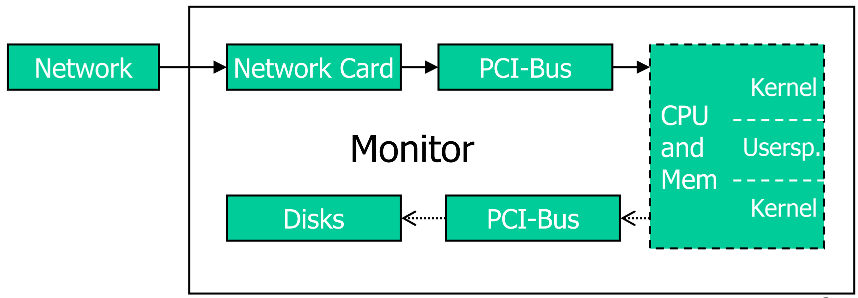
How to (passively) measure? Packet Monitoring

What to expect?

- Overview / What is packet monitor?
- How to acquire the data
- Handling performance bottlenecks
- Case Study: Packet Capture Performance
- Analyzing the transport and application layer
- (Mis-)Using the Bro Network Intrusion Detection System (NIDS) for network measurements

What is a Packet Monitor?

- Measuring / recording network data on a per packet basis
- Ordinary (although high-end) PC hardware
- Datapath:



Passive Monitoring: Challenges (1)

- User privacy & network security
- Data privacy vs. data sharing
- Data filtering
- Tap into live network traffic and extract packets
- Must not interfere with normal packet transmission
- Real-time: cannot control bandwidth, cannot postpone work

Passive Monitoring: Challenges (2) Performance Issues

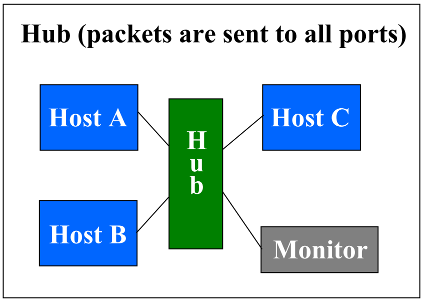
- High data rate
  - Bandwidth limits on CPU, I/O, memory, and disk/tape
  - Network cards optimized for bi-directional data transfer, not capturing
- High data volume
  - Space limitations in main memory and on disk/tape
  - Could do online analysis to sample, filter, & aggregate
- High processing load
  - CPU/memory limits for extracting, counting, & analyzing
  - Could do offline processing for time-consuming analysis
- General solutions to system constraints
  - Sub-select the traffic (addresses/ports, first *n* bytes)
  - Kernel and interface card support for measurement

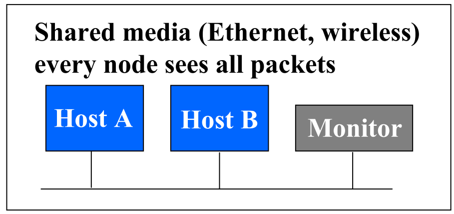
Monitoring Links: Overview

- How to get data off the network, without interfering normal transmission?
- For half-duplex:
  - Shared medium
  - Hub
- For full-duplex:
  - Monitor / SPAN port
  - "Tap"

Monitoring Links (1)

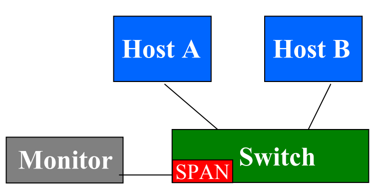
- Half Duplex: host cannot send and receive at the same time. Only one host can send.





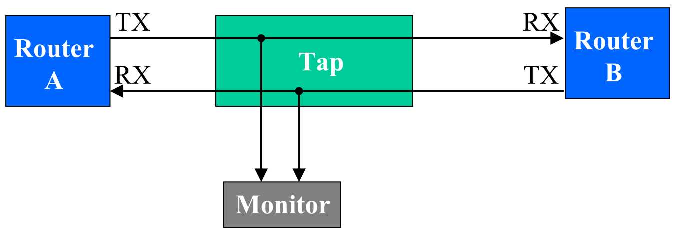
Monitoring Links (2)

- Full Duplex: host can send and receive at the same time
- Monitoring- / SPAN port on switch
  - every packet seen by switch copied to SPAN port
  - easy (every switch supports this)
  - all sending host are aggregated into one monitoring link ==> Packet loss



Monitoring Links (3)

- Full Duplex
- Tap into data
  - Only between two nodes (routers)
  - Can capture all traffic
  - Need two receive ports on Monitor
  - Fiber: purely optical
  - Copper: needs active components



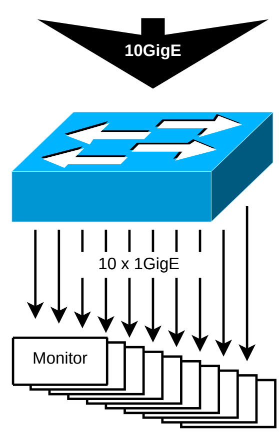
Handling Performance Bottlenecks

Handling high bandwidth

- Hard for monitors to cope with high load
  - Interrupt VS Polling
  - Long datapath => data is copied several times
- Use dedicated network monitoring cards
  - Often have several ports (for Taps)
  - Filtering / aggregation in hardware on card
  - Very expensive (EUR 3,500 for 1 Gpbs)
- Split traffic

Splitting traffic

- Problem: no recent host bus or disk system can handle the bandwidth needs for 10 Gbps
- Solution: split traffic and distribute the load (e.g., 10 Gbps on multiple 1 Gbps links)
  - Use a switch: link bundling features
  - Use specialized hardware
- Important: Keep corresponding data together
  - per IP, per IP-pair, per connection



Splitting Traffic: Link Bundling

- Etherchannel (Cisco switches) feature enables link-bundling for:
  - Higher bandwidth, redundancy
  - Or load-balancing, e.g., for Webservers
- Simple switches use only MAC addresses
  - => not suitable for router-to-router link
- On a Cisco 3750: any combination of IP and/or MAC addresses
  - => is sufficient for our scenario
- On Cisco 65xx: MACs, IPs and/or port numbers

Case Study: Packet Capture Performance

Goal:

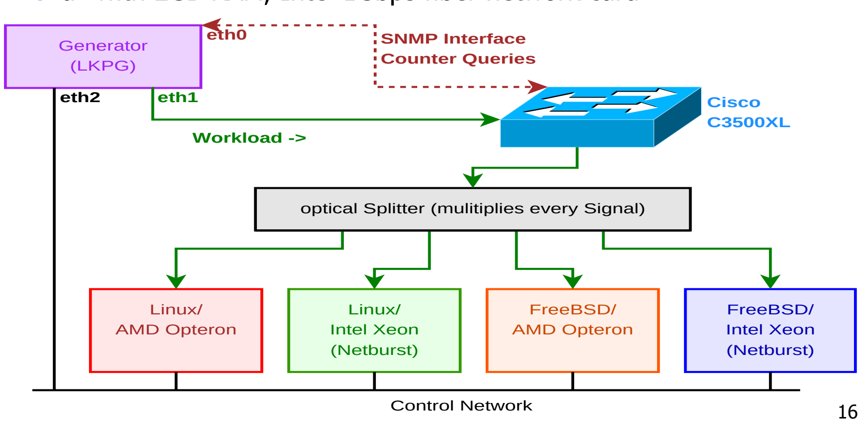
- See how measurement studies are conducted
- See what influence capture performance and what system is "best"

Case study: Packet Capture Perform.

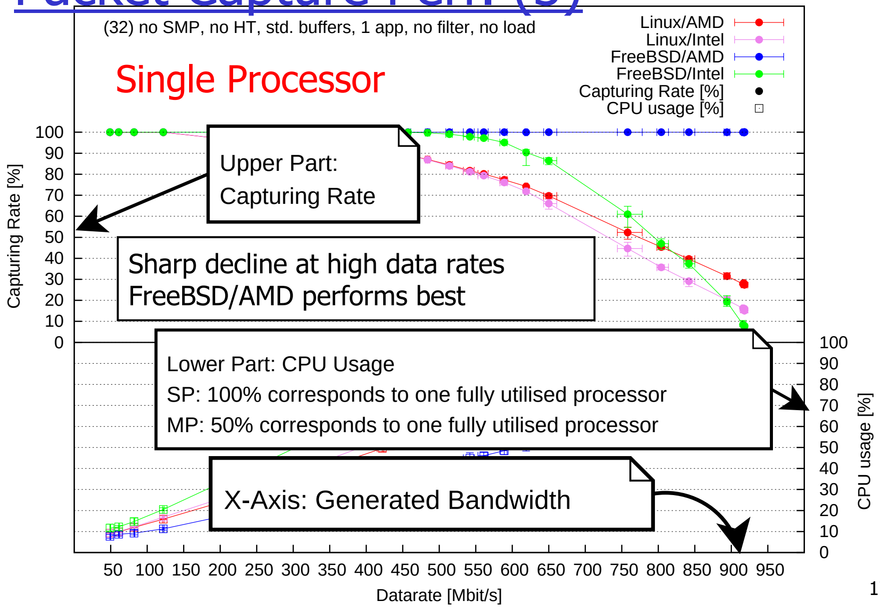
- Compare 1Gbps monitors based on standard hardware
  - Different CPU architectures, Different OSes
- Workload
  - Capture full packets, but do not analyze them
  - Identical input to all systems
  - Increase bandwidth until 1Gbps fully loaded
  - Realistic packet size distribution
- Metrics
  - Capturing rate: number of captured packets
  - System load: CPU usage while capturing

Packet Capture Perf. (2)

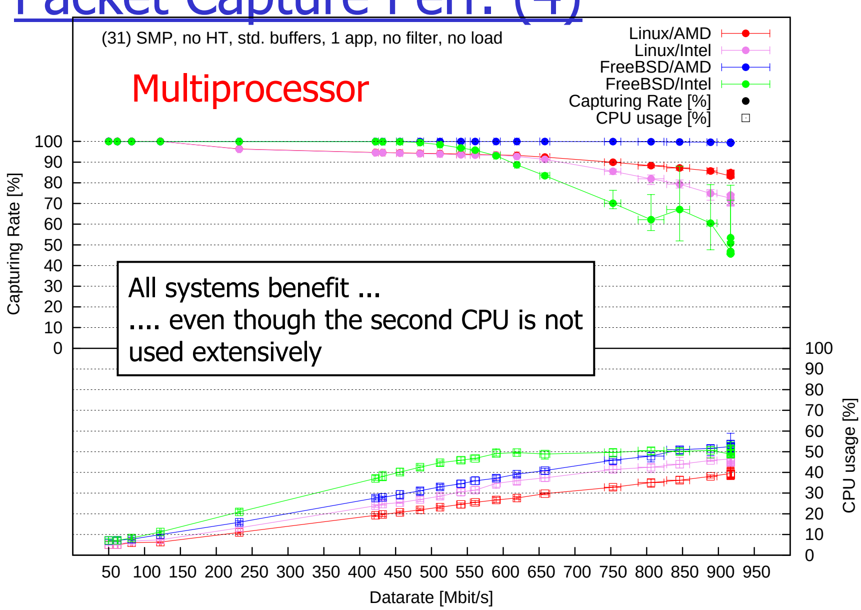
- Systems under Test:
  - AMD Opteron 244 and 270 VS. Intel Xeon
  - FreeBSD VS. Linux
  - all with 2GB RAM, Intel 1Gbps fiber network card



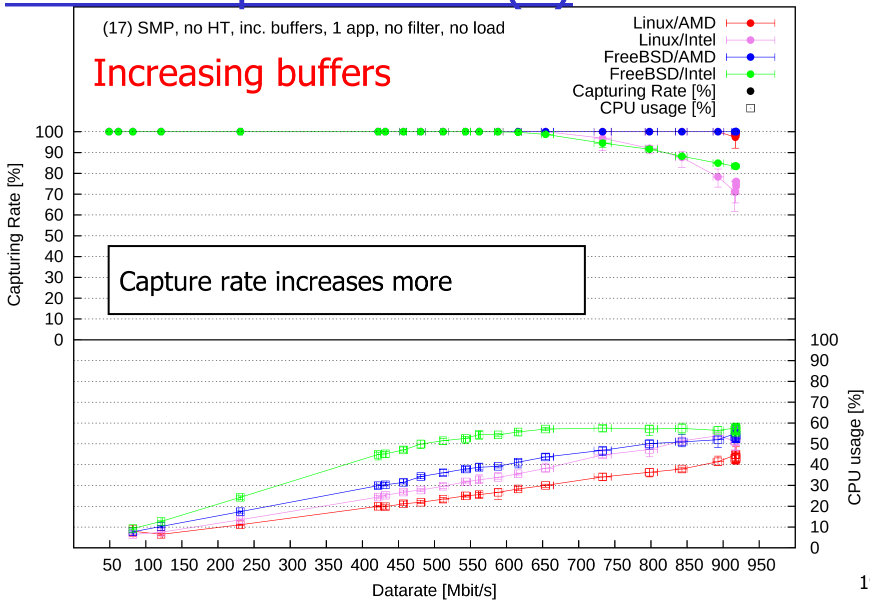
Packet Capture Perf. (3)



Packet Capture Perf. (4)



Packet Capture Perf. (5)



Packet Capture Perf. (6): Summary

- FreeBSD/AMD system performs best
  - Multiprocessor and increasing buffers help
- Additional insights
  - Filtering is cheap, even for large filter terms
  - Running multiple capture applications in parallel leads to bad performance
  - When using compression (e.g., ZIP) Intel has advantages
  - Intel Hyperthreading does not change performance

Analyzing the transport and application layer

How to get from packets to
connections (TCP, UDP) to
application level protocols (HTTP, DNS, etc.)

Packet VS Connections VS Applications

- Monitors deliver single packets
- Lots of measurements one can do on per packet basis
  - Timing, packet sizes, routing, IP stats, ...
- More measurements on transport layer (TCP/UDP)
  - Timing, connection size, ...
- But often we want to analyze application layer protocols (e.g., HTTP, SIP, etc.)

Application-level Messages

- Application-level transfer may span multiple packets
  - Demultiplex packets into separate "flows"
  - Identify by source/dest IP addresses, ports, and protocol
  - Maybe also application level identifiers

Application-level Messages: Reassembly

- Reconstructing ordered, reliable byte stream
  - De-fragment fragmented IP packets
  - Reassemble TCP segments
  - Sequence number and segment length in TCP header
  - Buffer to store packets in correct order & discard duplicates
- Packets might be missing (measurement drops)
- Packet might be truncated

Application-level Messages: Reassembly (2)

- Inconsistent retransmissions
  - TCP retransmission, but data does not match
- Need state per connection
- Idle connections
  - Is teardown missing?
  - Is there going to be more data?
  - Cannot keep state for ever (memory exhaustion)
  - => need strategy for state removal

Application-level Messages

Extraction of application-level messages

- Parsing the syntax of the application-level
  - Clients may not adhere to specification
- Identifying the start of the next message, e.g., HTTP
  - Absence of body
  - Presence of Content-Length
  - Chunk-encoded message
  - Multipart/byterange
  - End of TCP connection

(Mis-)Using the Bro Intrusion Detection System for Network Measurement

What is a NIDS?

- Network Intrusion Detection System (NIDS)
  - Monitors network traffic to detect attacks in real-time
  - Reports suspicious activity to operator
- A NIDS has to be robust
  - To protect itself from direct attacks
  - To detect / prevent evasions
  - Must be careful to not exhaust its resources (memory, CPU, disk)

Why use a NIDS for measurement?

- To perform its task a NIDS has to
  - De-Fragment IP packets
  - Reassemble TCP connections (and cope with inconsistencies and packet loss)
  - Keep track of connections, manage state
  - Track resource usage
  - Parse Application-layer protocols
  - Extract Application-layer messages and data elements
    - e.g., URLs, etc.
  - Handle broken protocol implementations
- All this things are also relevant to measurement

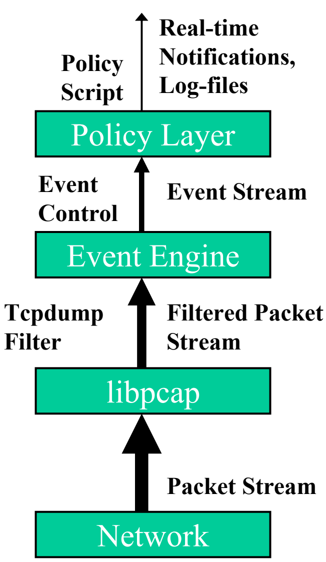
Bro: an open source NIDS

- Bro is open source
- Developed by Vern Paxson (UC Berkeley)
- Used as productive and research system
- Is modular and easy to extend
- We use it heavily
  - As NIDS to protect our network
  - Conduct NIDS research
  - Conduct network measurements

Bro System Philosophy

- Fundamentally, Bro provides a real-time network analysis framework
- Emphasis on
  - Application-level semantics
    - rare to analyze individual packets
  - Tracking information over time
    - Both within and across connections
    - Also archiving for later off-line analysis
- Strong separation of mechanism and policy
  - Much of the system is policy-neutral. I.e., no presumption of "good" or "bad"

Bro Architecture



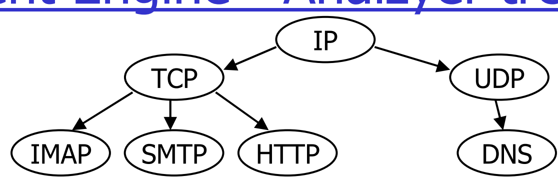
- "Policy Script" processes and aggregates event streams
- "Event engine" distills filtered stream into high-level network events

  E.g., **connection_attempt_seen, http_reply, user_logged_in**
- Kernel filters down high-volume stream via libpcap packet capture library
- Reads packets from network, sends up copy of all packets

Event Engine

- Event engine performs generic analysis
- Also termed the "Core"
- Written in C++
- Basic element of analysis is a "connection"
  - De-fragment IP
  - Reassemble TCP streams
  - Pass reassembled TCP (UDP) streams to application-level analyzers
  - Event engine uses an analyzer tree

Event Engine - Analzyer tree



- Tree elements can tune-out if not their protocol
- Data transforms as it flows through analyzers
  - E.g., packets -> byte stream -> lines of text
- As analyzers observe activity, they generate events
  - Events span several aggregation levels
  - All events triggered by a given packet executed before next packet is processed

Event Engine

- Events are basis of interface to policy script
- If no handler in policy script for given event, Event Engine skips work without generating event
- Writing Analyzers
  - Originally: Plain C++
  - New: BinPAC (= "Binary" Protocol Analyzer Compiler)
    - Declarative description of protocol (similar to BNF)
    - binpac tool generates C++
- Example events:

  new_connection, new_packet, icmp_echo_reply, http_header, authentication_rejected, ssh_server_version, ...

Policy Script Layer

- Bro specific scripting language
  - Procedural
  - Strong support for network data types (IPs, subnets, ports, hashtable, etc.)
  - Provides support for state management
- Receives (and processes) events from Event Engine
- Can also generate further events, that are handled by other Policy Scripts
- Tradeoff: where to implement functionality
  - Event Engine is fast; Script Layer is easy to implement

Advanced Bro Features

- Broccoli = Bro Client Communications Library
  - C interface for external programs to transmit & receive values and events
- Policy-level state management
  - Entries in hashtables, sets, etc. can time-out T sec. after creation / last write / last read
- Support for external analyses
  - Antivir, libmagic, GeoIP, passive OS fingerprinting...