Operational Experiences With High-Volume Network Intrusion Detection

Holger Dreger[1] Anja Feldmann[1] Vern Paxson[2]
Robin Sommer[1]

[1]TU München
Germany

[2]ICSI / LBNL
Berkeley, CA, USA

ACM Computer and Communications Security 2004

Network Intrusion Detection in High-Volume Networks

- Experience with open-source NIDSs in Gbps environments:
  - Snort dropped lots of packets ⇒ *CPU load too high*
  - Bro *additionally* consumed all memory ⇒ *stores too much state*
- Questions
  - Key factors in terms of resource usage?
  - Ways to reduce resource consumption?
  - Impact on detection rate?
- No answers available
  - Researchers often lack access to high-volume environments
  - Commercial vendors keep their techniques private

Outline

1. Environments
2. Operational Experiences
3. Extensions to the NIDS
4. Trade-Off: Detection Rate vs. Resource Usage

Environments

- Operational environments
  - Munich Scientific Network
  - University of California, Berkeley
  - Lawrence Berkeley National Laboratory
- Main research environment: Munich Scientific Network
  - Two major universities and several research institutes
  - Gbps Internet uplink transferring 1-2 TB each day
  - 50,000 hosts; 65,000 users
  - Monitor: Dual Athlon 1800+, FreeBSD 5.2.1
- Traces augment our study to demonstrate challenges

Bro NIDS

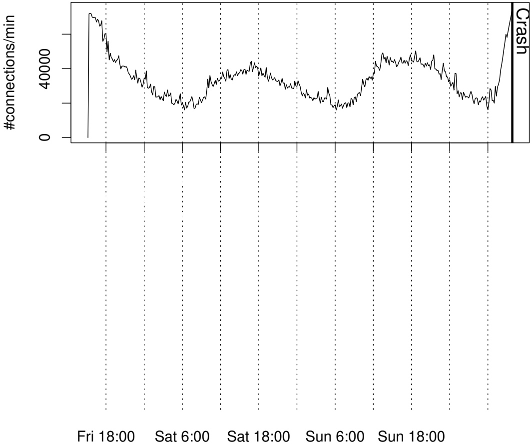
- Powerful open-source NIDS
- Research project started in 1995
- Supports different approaches to intrusion detection
- Focuses on
  - Semantically high-level analysis
  - Efficiency
  - Extensibility
  - Resistance to evasion
  - Separation of mechanism and policy

Outline

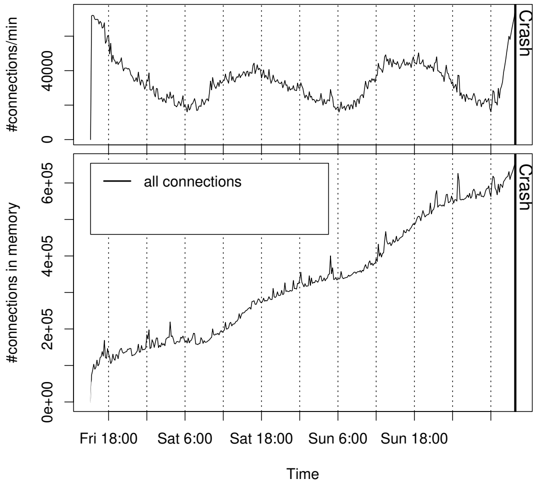
Memory Consumption

- *Stateful* NIDS maintains representation of network's state
  - The more it knows about the network the more it can detect
- Connection state
  - Instantiated when connection starts
  - Removed when connection ends
- User state
  - NIDS may provide scripting language for customizations
  - Data structures store state (e.g., arrays)
  - User is responsible to delete state eventually

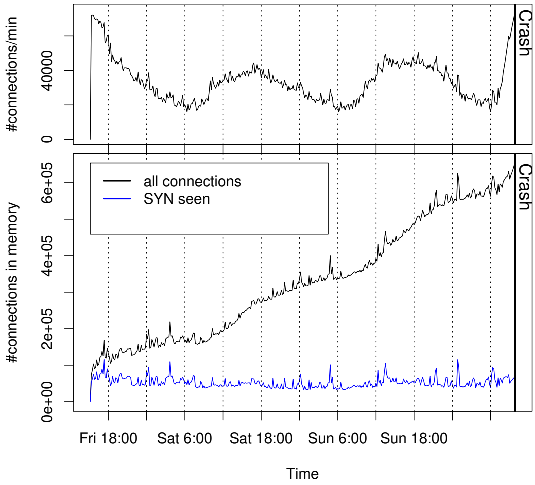
Running Out Of Memory: Connection State



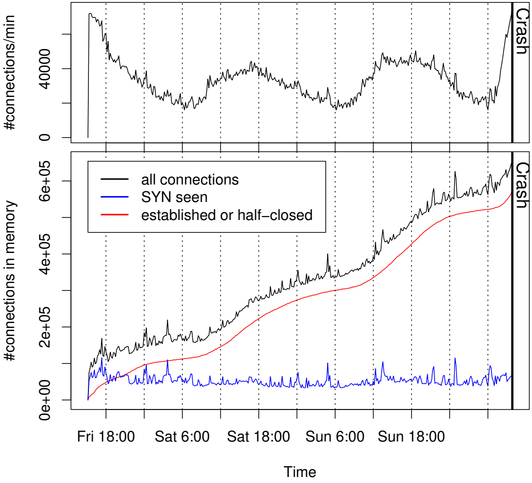
Running Out Of Memory: Connection State



Running Out Of Memory: Connection State



Running Out Of Memory: Connection State



Observation

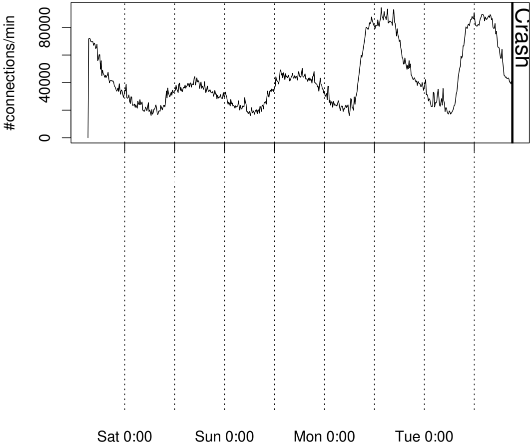
Many established connections are not deleted

Running Out Of Memory: Connection State

- Avoiding evasion is design goal
  - Only delete connection state when it is *safe*
- Problem
  - Not feasible in high-volume environments
- Approaches to expire connection prematurely
  - Limit number of connections in memory
  - Limit total amount of connection state memory
  - *Limit connection life-time with inactivity time-outs*
- Trade-Off
  - Memory-consumption vs. detection rate

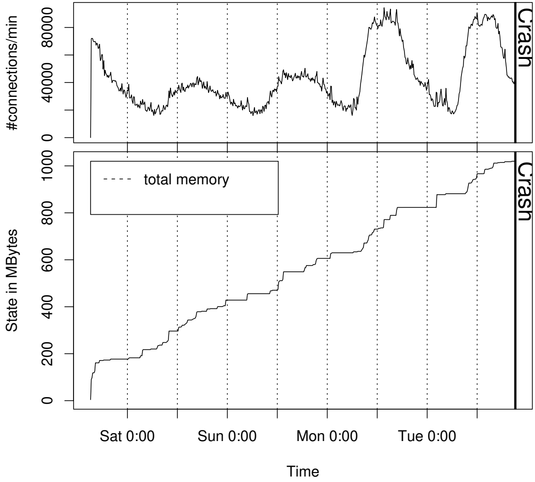
Running Out of Memory: User State

- Bro's *scan detector* is a user-level script



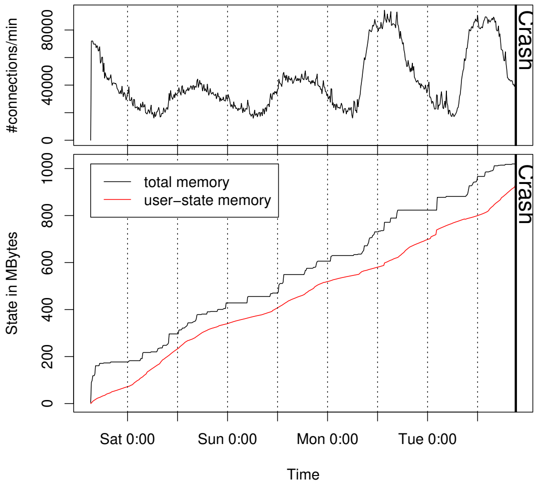
Running Out of Memory: User State

- Bro's *scan detector* is a user-level script



Running Out of Memory: User State

- Bro's *scan detector* is a user-level script



Observation

Much of the user state is not deleted

Running Out of Memory: User State

- Avoiding evasion is design goal
  - Detecting *all* scans requires remembering all connections
- Problem
  - Again not feasible in high-volume environments
- Added mechanisms to expire user state
  - Ease deleting state explicitly
  - Allow deleting state implicitly via time-outs
- Adapted default scripts to make use of them

Outline

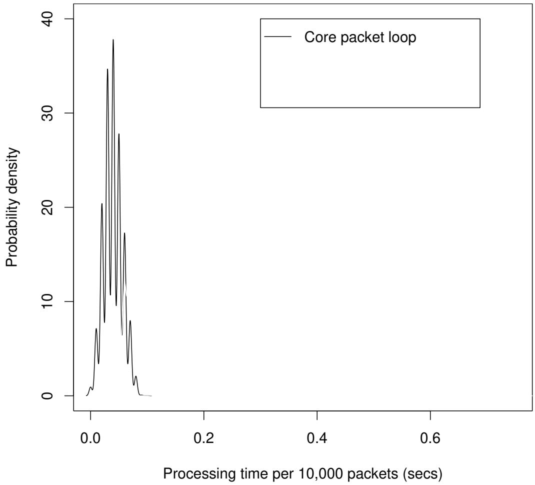
1. Environments
2. Operational Experiences
   - Memory Consumption
   - CPU Consumption
3. Extensions to the NIDS
4. Trade-Off: Detection Rate vs. Resource Usage

CPU Consumption

- When analysis exceeds available time packet drops occur
- Major reason: network load exceeds processing capacity
  - Current commodity hardware cannot analyze every packet
  - Need to find a tractable subset of traffic
- Problem: Internet traffic is very dynamic
  - Long-term effects: time-of-day and day-of-week
  - Short-term effects: traffic is multi-fractal
  - Anomalies: worms, floods, misbehaving software
- Hard to predict time even for well-understood traffic
  - Per-packet processing time varies widely
  - Processing spikes triggered by individual packets

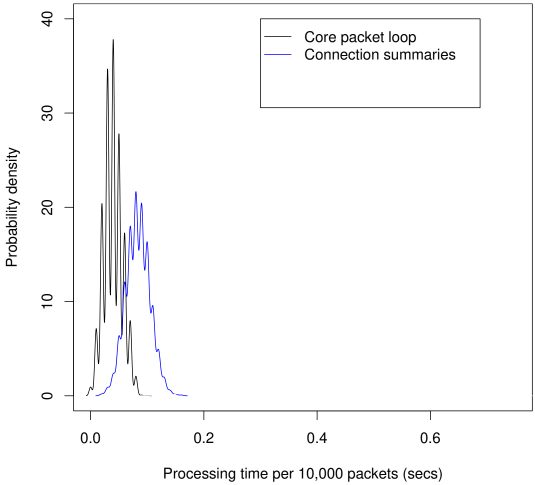
Fluctuating Processing Times

- Example: Running times for different depths of analysis



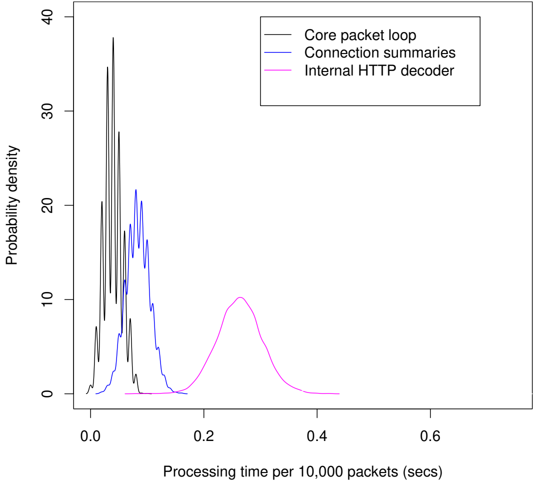
Fluctuating Processing Times

- Example: Running times for different depths of analysis



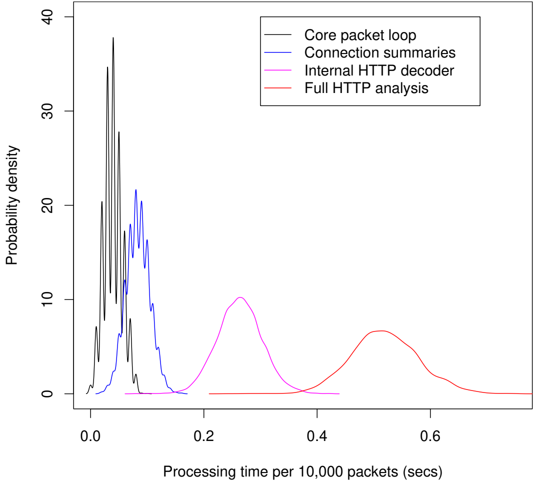
Fluctuating Processing Times

- Example: Running times for different depths of analysis



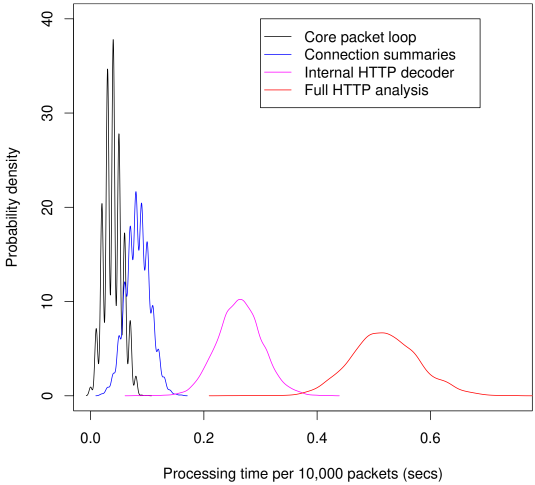
Fluctuating Processing Times

- Example: Running times for different depths of analysis



Fluctuating Processing Times

- Example: Running times for different depths of analysis



Observation

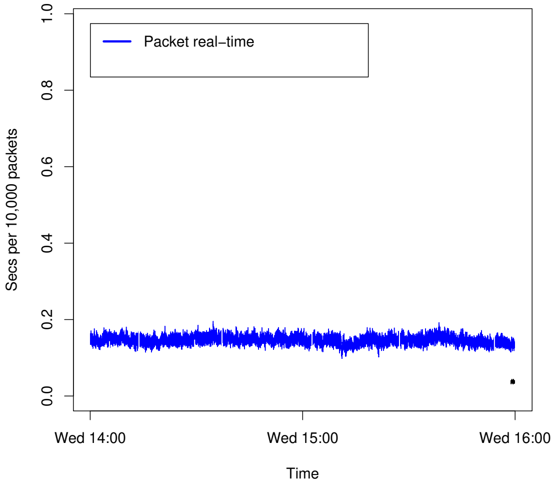
Per-packet time fluctuates a lot

Consequence

Hardly possible to predict worst-case

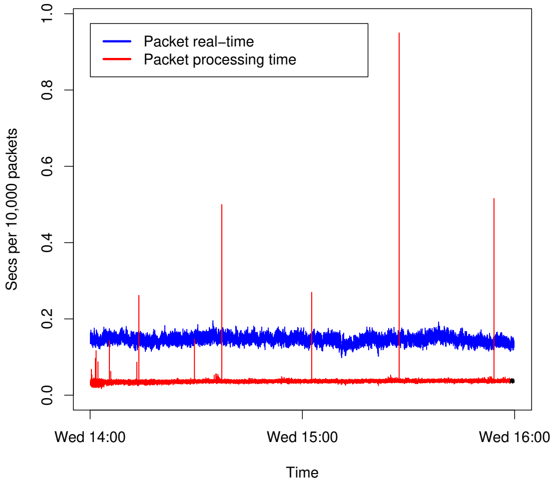
Processing Spikes

- Example: Spikes triggered by a single packet



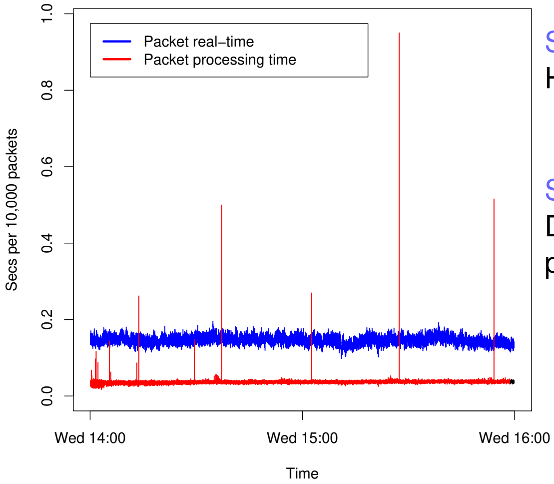
Processing Spikes

- Example: Spikes triggered by a single packet



Processing Spikes

- Example: Spikes triggered by a single packet



Spikes
Hash table resizes

Solution
Distribute across many packets

Outline

1. Environments
2. Operational Experiences
3. Extensions to the NIDS
4. Trade-Off: Detection Rate vs. Resource Usage

High-Volume Extensions

- New time-outs
  - Automatically expire internal and user state
- Connection compressor
  - Defers instantiation of connection state
- Load-levels
  - Adapt the NIDS's configuration to the current network load
  - Measure load by either CPU usage or packet drops
- Flood-detector
  - Excludes flood victim from analysis

Trade-Off: Detection Rate vs. Resource Usage

- Usual trade-off in computer science
  - Time vs. memory
- Network Intrusion Detection
  - Detection rate vs. resource usage
- Bro's design emphasizes detection
- High-volume environments require different trade-off
- Trade-off is policy decision left to the user
- Variant of Kerkhoff's principle avoids predictability
  - Detection mechanisms are public
  - Environment-specific parameterizations are private

Summary

- Network intrusion detection in high-volume environments
  - Unusual trade-off between detection rate and resource usage
  - Dynamic traffic makes it hard to find a stable point of operation
- Our work
  - Thorough understanding of the trade-off
  - Tuning mechanisms to successfully operate the system
- Outlook
  - Deploying specialized monitoring hardware
  - Refining measurement models
  - Developing auto-configuration tool
  - Adapting to still larger link capacities

Operational Experiences With High-Volume Network Intrusion Detection

Holger Dreger[1] Anja Feldmann[1] Vern Paxson[2]
Robin Sommer[1]

[1]TU München
Germany

[2]ICSI / LBNL
Berkeley, CA, USA

ACM Computer and Communications Security 2004

Further Issues In High-Volume Networks

- Artifacts of the monitoring environment
  - Limits imposed by commodity PC hardware
  - Merging of multiple Gbps into one
  - Router-side buffer overruns
  - Optical-taps: uni-directional

Further Issues In High-Volume Networks

- Artifacts of the monitoring environment
  - Limits imposed by commodity PC hardware
  - Merging of multiple Gbps into one
  - Router-side buffer overruns
  - Optical-taps: uni-directional
- Programming deficiencies will be severely punished
  - Expecting any sort of "reasonable" traffic is sure to fail
  - Memory leaks are a major hassle