An Overview of Zeek Performance

Vern Paxson

Corelight, Inc.
EECS Department, University of California, Berkeley
International Computer Science Institute

vern@{corelight.com,berkeley.edu}

Oct 14, 2020
Thinking About Zeek Performance

• From its initial design, Zeek meant to provide high-performance network monitoring

• Overall performance depends on aggregate perf. of multiple (quite different) components
  – Slowest of these imposes performance limit

• This talk:
  – Performance implications of Zeek’s architecture
    • And how that has evolved
  – Current work targeting one component (scripting)

• Goal: provide useful mental models
Original Architecture

- Taps network link passively, sends up a copy of all network traffic.
• Kernel filters down high-volume stream via standard libpcap packet capture library.

Filtering vital for managing processing load
• “Event engine” decodes protocols, distills filtered stream into high-level, policy-neutral events reflecting underlying network activity
  – E.g., connection_attempt, http_reply, file_hash, pe_dos_code
  – These span a range of semantic levels
  – Currently ~650 different types
Original Architecture

- Scripts (written in custom Zeek scripting language) process event stream, incorporating:
  - Context/state from past events
  - Additional input sources
  - Site’s particular policies

  … and take action:
  - Export extensive logs
  - Generate real-time alerts ("notices")
  - Execute programs as a form of response

Today Zeek ships with 48,000 lines of scripts
Architecture As It Has Evolved

Scalable high performance via Zeek Cluster. Requires packet broker or load-balancing NIC.
Architecture As It Has Evolved

- **Policy Script Interpreter**
- **Event Stream**
  - **Event Control**
  - **Tcpdump Filter**
  - **Filtered Packet Stream**
- **Packet Stream**

**Network**

- **libpcap**
- **Real-time Action Log Archive**

No **static** filtering by default; analyzes off-port traffic using *Dynamic Protocol Detection*.
Dynamic filtering facilitates very high-speed monitoring using *shunting*.
Architecture As It Has Evolved

Extensive library functionality (450 built-in-functions), input/logging/output & analysis frameworks
Architecture As It Has Evolved

Extensibility via **packages**: plug-ins (C++ code) & scripts (150 in packages.zeek.org)
Summary of Zeek Performance

• Processing mainly spent in 4 areas:
  1. Packet dispatch / dynamic filtering (NIC, OS)
  2. Event engine (C++)
     • Including plug-in components
  3. Script interpreter (C++ recursing over trees)
     • Including package components
  4. Built-in functions called by scripts (C++)
     • Libraries, plug-ins
• Can we speed up these scripts?
E.g: Zeek Script Representation

```zeek
function demo(a: int, b: int): int {
    if (a * a > b) {
        return a * a + b;
    } else {
        return b * b - a;
    }
}
```

Function has two arguments, `a` and `b`
function demo(a: int, b: int): int
{
    if ( a * a > b )
        return a * a + b;
    else
        return b * b - a;
}
E.g: Zeek Script Representation

```plaintext
function demo(a: int, b: int): int {
    if ( a * a > b )
        return a * a + b;
    else
        return b * b - a;
}
```

“demo” returns an integer value
function demo(a: int, b: int): int {
    if (a * a > b) {
        return a * a + b;
    } else {
        return b * b - a;
    }
}

Zeek turns function description into a tree with 18 elements
Upon a call to `demo()`, Zeek Interpreter assigns arguments to local variables ‘a’ and ‘b’ …  
… and then evaluates each element in the tree as needed.
Zeek Interpreter Execution*

Including executing the true or false branch, this works out to:
12 C++ function/method calls
16 memory management increments & decrements, and
4 "Val" creations/destinations
Speeding Up Script Execution

• Core idea: further compile script “trees” to *low-level abstract machine* (ZAM)
  – Like interpreted assembly language
• Using low-level interpretation *avoids* need for complex tool chain
  – (changing a script requiring recompiling/relinking)
• Gaining speed also requires:
  – A range of *optimizations*
  – Changing the *low-level representation* of Zeek script values (avoid some memory management)
“Range of Optimizations”

• Identify repeated computations (like “a*a” in `demo()` example), *compute-once-and-reuse*

• Identify expressions with constants, *compute result at compile-time*

• Remove values computed-but-*not*-used

• Double up variables whose “lifetimes” don’t overlap
  – Improves memory management

• **Inline** function calls to avoid overhead, identify opportunities for “constant propagation”
function demo(a: int, b: int): int {
    if (a * a > b) {
        return a * a + b;
    } else {
        return b * b - a;
    }
}

Zeek turns function description into a *tree* with 18 elements
function demo(a: int, b: int): int
{
    if ( a * a > b )
        return a * a + b;
    else
        return b * b - a;
}
Zeek Interpreter Execution*

Including executing the true or false branch, this works out to:
12 C++ function/method calls
16 memory management increments & decrements, and
4 “Val” creations/destructions
Zeek Optimized Execution

0: load-val-VV 1 (a), interpreter frame[0]
1: load-val-VV 2 (b), interpreter frame[1]
2: times-VVV-I 4 (tmp), 1 (a), 1 (a)
3: lt-VVV-I-cond 2 (b), 4 (tmp), 6
4: add-VVV-I 6 (tmp), 4 (tmp), 2 (b)
5: return-V 6 (tmp)
6: times-VVV-I 5 (b), 2 (b), 2 (b)
7: sub-VVV-I 3 (a), 5 (b), 1 (a)
8: return-V 3 (a)

Including executing the true or false branch, this works out to: 1 C++ function/method call; 6-7 loop+switch iterations 0 memory management increments & decrements, and 1 “Val” creations/destructions (0 if inlined)
Status of Script Optimization

• Main work done
  – topic/vern/script-opt
  – About 24,000 LOC (> entire original impl.!!)
  – Needs review, updates for recent Zeek changes
  – Needs volunteers!

• Targeting inclusion in Zeek 4.0 (experimental)

• How much faster is it?
Summary of Zeek Performance

- Processing mainly spent in 4 areas:
  1. Packet dispatch / dynamic filtering (NIC, OS)
  2. Event engine (C++)
     - Including plug-in components
  3. Script interpreter (C++ recursing over trees)
     - Including package components
  4. Built-in functions called by scripts (C++)
     - Libraries, plug-ins

This part is much faster: 3-10x.
Summary of Zeek Performance

• Processing mainly spent in 4 areas:

1. Packet dispatch / dynamic filtering (NIC, OS)
2. Event engine (C++)
   • Including plug-in components
3. Script interpreter (C++ recursing over trees)
   • Including package components
4. Built-in functions called by scripts (C++)
   • Libraries, plug-ins

This part is much faster: 3-10x. But *these* parts remain.
Summary of Zeek Performance

• Processing mainly spent in 4 areas:
  1. Packet dispatch / dynamic filtering (NIC, OS)
  2. Event engine (C++)
     • Including plug-in components
  3. Script interpreter (C++ recursing over trees)
     • Including package components
  4. Built-in functions called by scripts (C++)
     • Libraries, plug-ins

This part is much faster: 3-10x.
But these parts remain.
Actual operational gain depends on traffic & config.
**However:** can now be more fearless using heavy scripting!
Stay Connected

Website - Zeek.org

Mailing List - zeek@lists.zeek.org


Find out more ways to connect at: https://zeek.org/community/