ETTM:
A Scalable, Fault-tolerant Network Manager

Colin Dixon

Hardeep Uppal
UMass Amherst

Vjekoslav Brajkovic, Dane Brandon,
Thomas Anderson and Arvind Krishnamurthy
University of Washington
What I want from a network

- Services
  - NAT (v4↔v4, v4↔v6 or both)
  - Firewall
  - Traffic Prioritization
  - Transparent Web Cache
  - Remote Access (VPN)
  - Pervasive Intrusion Detection (IDS/DPI)

- Requires buying several proprietary middleboxes

- Even that is potentially inconsistent, inextensible, and doesn’t scale
Intrusion detection today

- Deployed at the edge of the network
- Difficult to catch internal threats
- Increased deployment requires coordination
A different approach

- An End to the Middle (ETTM)

- We do not need physically centralized hardware to make logically centralized decisions

- Implement network functionality as “virtual middleboxes” running as software on users’ PCs
Pervasive intrusion detection

- Assume a trusted shim at each host to provide monitoring and control
- Gives pervasive monitoring and the ability to quarantine hosts
## Assumptions

- Enterprise Network
- Single administrative domain

- Hosts have trusted computing hardware
- Hosts are multicores
- Hosts are virtualized
- Switches provide access control

Increasingly true of commodity PCs

802.1X & OpenFlow
Goals of ETTM

- Make it easy to extend network functionality
  - One platform with simple APIs

- Automatically scale with network demand
  - Run management on already-present end-hosts

- Function pervasively, consistently, fault-tolerantly
  - Distributed consensus
Outline

- Introduction & Motivation
- ETTM
  - Design
  - Evaluation
- Conclusion
Challenges

• Can users’ commodity PCs be trusted to faithfully carry out network management tasks?

• Can users’ commodity PCs quickly make consistent decisions that survive failures?

• How can we enable developers and administrators to build and deploy new features?
Challenges

- Can users’ commodity PCs be trusted to faithfully carry out tasks?
- Can users’ commodity PCs quickly make consistent decisions that survive failures?
- How can we enable developers and administrators to build and deploy new features?
Trusted Authorization

Step 1: Switch discovers new end-host and sends EAP Request Identity
- Step 1: 50 ms

Step 2: End-host responds with EAP Response with public key to use
- Step 2-4: 20 ms

Step 3: Message tunneled to verification server using 802.1X
- Step 5: 1 s

Step 4: Verification server sends challenge nonce to end-host
- Step 6: varies

Step 5: End-host sends integrity measurements
- Step 7: 20 ms

Total: 1.09s (Can be overlaid with commodity OS boot)
Legacy hosts

• What about hosts which don’t (or can’t) run their own AEE?

• AEEs are virtual and need not run on the same host they are filtering
Fault tolerance/Consistency

- **Shared Table**
  - 1 row per application
  - Each row is a log of state updates
  - Each cell is the result of a Paxos round

- **Catastrophic Failures**
  - Allow unsafe progress
  - Fork row, merge changes after the fact

<table>
<thead>
<tr>
<th>Name</th>
<th>1</th>
<th>2</th>
<th>3 …</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAT</td>
<td>A→B</td>
<td>C→D</td>
<td>…</td>
</tr>
<tr>
<td>unsafe NAT</td>
<td>from NAT at t=4</td>
<td>E→F</td>
<td>…</td>
</tr>
<tr>
<td>Web Cache</td>
<td>host down</td>
<td>host up</td>
<td>…</td>
</tr>
<tr>
<td>Topology</td>
<td>link up</td>
<td>link dn</td>
<td>…</td>
</tr>
<tr>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
</tr>
</tbody>
</table>
μvrouter

- Runs in the AEE, invokes filters on packets
- Open vSwitch implementation
  - Line-speed
  - Accepts OpenFlow commands
  - Limited to packet header operations
- iptables implementation
  - 250 Mbps
  - Full C++ filters
  - Access to complete packet
Application Filters

- μvrouter determines which filters to invoke on each packet and in what order
- Filters capture, delay, rewrite, drop and inject packets
- μvrouter also calls filters’ upkeep functionality, fetches any new packets

- matchOnHeader(): true if the filter works on only headers
- getPriority(): returns integer used to order filters
- getName(): returns user-friendly name
- matchHeader(ipH, tcpH, udpH): check if the filter matches headers
- match(pkt): check if the filter matches full payload
- filter(pkt): processes a packet, returns an action
- upkeep(): called periodically for maintenance
- getReadyPackets(): used to inject any new packets
Putting it all together

| Name | 1       | 2       | 3       | 4       | ...
|------|---------|---------|---------|---------|---------
| NAT  | A→B     | C→D     |         |         | ...
|      |         |         |         |         | ...

Hypervisor w/TPM

Attested Exec. Env.

Commodity OS

Application

Filter

packet

NAT filter

Applications

Filters/µrouter

Distributed Consensus

packet

Name

1

2

3

4 ...

packet

packet

packet

packet

packet

packet

packet
Outline

- Introduction & Motivation
- ETTM
  - Design
  - Evaluation
- Conclusion
Implemented ETTM Apps

- NAT
- Deep Packet Inspection/IDS
- Web Cache
- Traffic Prioritization
- Worm Scan Detection
- Firewall

- Most are only a few 100s of lines
Is consensus fast?

We make fault-tolerant decisions in under 1 ms on a wired LAN.
Does consensus scale?

We can add between 1700 and 8000 cells to a row each second.
Do filters hurt performance?

For reasonable flows (>10 KB), performance is identical

Achieve 1 Gb/s
ETTM Bandwidth Allocation

Latency to load google.com

Latency spikes with BitTorrent flow

Latency with bandwidth reservations

Bandwidth allocator keeps latency stable
Conclusion

- Enterprise network management today is complex, expensive and ultimately unsatisfying

- We propose a new approach: implementing network management tasks as software on end-hosts
  - Easy to write and distribute network apps
  - Management resources scale with demand
  - Tasks run pervasively, consistently and tolerate faults

- Implemented NAT, IDS, Cache, BW Alloc., Scan Det.