An Overview of P4 Programmable Switches and Applications to Cybersecurity and Networks

Jorge Crichigno
College of Engineering and Computing, University of South Carolina
jcrichigno@cec.sc.edu
http://ce.sc.edu/cyberinfra

IEEE International Conference on Telecommunications and Signal Processing (TSP)

Online
July 12-14, 2023
Agenda

• Introduction to P4 Programmable Switches
  ➢ Legacy devices and Software-defined Networking (SDN)
  ➢ Programmable data plane (PDP) switches

• Dynamic Router’s Buffer Sizing using P4 Switches
  ➢ Buffer sizing problem
  ➢ A passive application using P4 switches

• DGA Family Classification using DNS Deep Packet Inspection on P4 Switches
  ➢ Domain Generation Algorithms (DGA) used by malware’s command and control (C2)
  ➢ An application for detection and classification of DGAs using P4 switches

• Conclusion
Introduction to P4 Programmable Switches
Traditional (Legacy) Networking

• Since the explosive growth of the Internet in the 1990s, the networking industry has been dominated by closed and proprietary hardware and software

• The interface between control and data planes has been historically proprietary
  ➢ Vendor dependence: slow product cycles of vendor equipment, no innovation from end programmers
  ➢ A router is a monolithic unit built and internally accessed by the manufacturer only
Software-Defined Networking (SDN)

- Protocol ossification has been challenged first by SDN
- SDN (1) explicitly separates the control and data planes, and (2) enables the control plane intelligence to be implemented as a software outside the switches by end programmers
- The function of populating the forwarding table is now performed by the controller
**SDN Limitation**

- SDN is limited to the OpenFlow specifications
  - Forwarding rules are based on a fixed number of protocols / header fields (e.g., IP, Ethernet)
- The data plane is designed with fixed functions (hard-coded)
  - Functions are implemented by the chip designer
Can the Data Plane be Programmable?

• Evolution of the computing industry

P4 Programmable Switches

- P4\(^1\) programmable switches permit **end programmers** to program the data plane
  - Define and parse new protocols
  - Customize packet processing functions
  - Measure events occurring in the data plane with high precision
  - Offload applications to the data plane

\(^1\) P4 stands for stands for Programming Protocol-independent Packet Processors
P4 Programmable Switches

- P4\(^1\) programmable switches permit **end programmers** to program the data plane
  - Define and parse new protocols
  - Customize packet processing functions
  - Measure events occurring in the data plane with high precision
  - Offload applications to the data plane

Reproduced from N. McKeown. Creating an End-to-End Programming Model for Packet Forwarding. Available: [https://www.youtube.com/watch?v=fIBua06YZI0&t=631s](https://www.youtube.com/watch?v=fIBua06YZI0&t=631s)
Generalized Forwarding: Match + Action

- Each switch contains table/s
  - Match bits in arriving packet (match phase)
  - Take action - Many header fields can determine the action (action phase)
    - Drop
    - Copy
    - Modify
    - Forward (destination-based forwarding is just a particular case)
    - ...
Dynamic Router’s Buffer Sizing using Passive Measurements and P4 Programmable Switches
Buffer Sizing Problem

- Routers and switches are designed to include packet buffers
- The size of buffers imposes significant implications on the performance of the network
- If the buffer allocated to an interface is
  - Very large, then packets may experience excessive delay ("bufferbloat")
  - Very small, then there may be a large packet drop rate and low link utilization
Buffer Sizing Problem

• General rule-of-thumb in the 90s was that the buffer size must equal the Bandwidth-delay product (BDP)

  ➢ Buffer = C * RTT
  ➢ C is the capacity of the port and RTT is the average round-trip time (RTT)

• The “Stanford rule” corrected the previous rule

  ➢ Buffer = (C * RTT)/√N
  ➢ N is the number of long (persistent over time) flows traversing the port

• Operator hardcodes the buffer size based on the typical traffic pattern
Proposed System

- The buffer size is dynamically modified
- A P4 switch is deployed passively to compute:
  - Number of long flows
  - Average RTT
  - Queueing delays
  - Packet loss rates
- The control plane sequentially searches for a buffer that minimizes delays and losses
- The searching algorithm is Bayesian Optimization (BO) with Gaussian Processes

---

Proposed System

- The buffer size is dynamically modified
- A P4 switch is deployed passively to compute:
  - Number of long flows
  - Average RTT
  - Queueing delays
  - Packet loss rates
- The control plane sequentially searches for a buffer that minimizes delays and losses
- The searching algorithm is Bayesian Optimization (BO) with Gaussian Processes

---

Proposed System

- Note that the system incorporates:
  - Customized packet processing
  - Nanosecond resolution measurements
  - Per-packet visibility
- The P4 apps run on the PDP chip at line rate

---

Evaluation

- 1000 senders
- P4 switch: Wedge100BF-32X with Intel’s Tofino ASIC
- Legacy router: Juniper router MX-204
- Different congestion control algorithms
- Access network:
  - $C_1 = 40\text{Gbps}$, $C_2 = 1\text{Gbps}$
- Core network:
  - $C_1 = 10\text{Gbps}$, $C_2 = 2.5\text{Gbps}$
Results

- Combined metric accounting for packet loss and delay $[0, 1]$ (the lower, the better)
- Top heatmaps: access network
- Bottom heatmaps: core network
- The Mixed scenario combines multiple congestion control algorithms

<table>
<thead>
<tr>
<th></th>
<th>Tiny</th>
<th>Stanford</th>
<th>BSCL</th>
<th>BDP</th>
<th>Bloated</th>
<th>ADT</th>
<th>P4BS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vegas</td>
<td>0.13</td>
<td>0.24</td>
<td>0.43</td>
<td>0.78</td>
<td>0.08</td>
<td>0.20</td>
<td>0.39</td>
</tr>
<tr>
<td>NewReno</td>
<td>0.13</td>
<td>0.23</td>
<td>0.48</td>
<td>0.99</td>
<td>0.09</td>
<td>0.19</td>
<td>0.43</td>
</tr>
<tr>
<td>Cubic</td>
<td>0.08</td>
<td>0.13</td>
<td>0.34</td>
<td>0.79</td>
<td>0.03</td>
<td>0.09</td>
<td>0.33</td>
</tr>
<tr>
<td>Illinois</td>
<td>0.44</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>0.14</td>
<td>0.34</td>
<td>1.00</td>
</tr>
<tr>
<td>BBvR2</td>
<td>0.23</td>
<td>0.30</td>
<td>0.56</td>
<td>1.00</td>
<td>0.19</td>
<td>0.27</td>
<td>0.62</td>
</tr>
<tr>
<td>Mixed</td>
<td>0.23</td>
<td>0.46</td>
<td>1.00</td>
<td>1.00</td>
<td>0.16</td>
<td>0.34</td>
<td>0.93</td>
</tr>
</tbody>
</table>

Results

- 100 VoIP calls playing 20 reference speech samples (G.711.a)
- The Perceptual Evaluation of Speech Quality (PESQ) compares an error-free audio signal to a degraded one (the higher, the better)
- The z-score considers both the delay and the PESQ (the higher, the better)
Results

• These results use real traffic traces from CAIDA\(^1\) and MAWI\(^2\)
• They include long and short flows
• P4BS found a balance such that:
  ➢ The FCT of short flows is close to that of the Stanford buffer
  ➢ The FCT of long flows is close to that of the bloated buffer

---

\(^1\) Center for Applied Internet Data Analysis (CAIDA). [https://www.caida.org/](https://www.caida.org/)
DGA Family Classification using DNS Deep Packet Inspection on P4 Programmable Switches
Introduction to DGAs

• Attackers often use a Command and Control (C2) server to establish communication between infected host/s and bot master

• Domain Generation Algorithms (DGAs) are the *de facto* dynamic C2 communication method used by malware, including botnets, ransomware, and many others
Introduction to DGAs

• DGAs evade firewall controls by frequently changing the domain name selected from a large pool of candidates.

• The malware makes DNS queries to resolve the IP addresses of these generated domains.

• Only a few of these queries will be successful; most of them will result in Non-Existent Domain (NXD) responses.

(1) DNS queries. (2) (NXD) replies. (3) Eventually, a query for the actual domain is sent and malware-C2 communication starts.
Introduction to DGAs

- DGAs evade firewall controls by frequently changing the domain name selected from a large pool of candidates.
- The malware makes DNS queries to resolve the IP addresses of these generated domains.
- Only a few of these queries will be successful; most of them will result in Non-Existent Domain (NXD) responses.
Existing Mitigation Techniques

- **Context-aware** approaches analyze the **network traffic** behavior to fingerprint DGAs
  - Slow since they typically analyze batches of traffic offline

- **Context-less** approaches analyze domain names (**DNS-based**) via ML models
  - The use of a general-purpose CPU/GPU may create a bottleneck due to high traffic volume

- There is a need for a system that
  - uses both context-aware and context-less features
  - detects and classifies DGAs based on the family (Trojan, backdoor, etc.)
Proposed System

• The P4 PDP switch collects and stores the **context-aware (traffic) features** of the hosts
  - Number of IP addresses contacted
  - Inter-arrival Time (IAT) between consecutive IP packets
  - Number of DNS requests made
  - Time it takes for the first NXD response to arrive
  - IAT between subsequent NXD responses

---

**Proposed System**

- When an NXD response is received, the switch performs DPI on the domain name to extract **context-less (domain) features**
  - The switch sends the collected features to the control plane
  - The control plane runs the intelligence to classify the DGA family and initiate the appropriate incidence response

---

Proposed System

- The scheme uses the bigram technique for **context-less (domain)** analysis:
  - It computes the bigram of the domain name; a bigram model may suffice to predict whether a domain name is a legitimate human readable domain.

\[
\text{score} \ (d) = \sum_{\forall \ subdomain \ s \in d} \left( \sum_{\forall \ bigram \ b \in s} f_s^b \right) \quad \text{Where} \ f_s^b \ \text{is the frequency of the bigram} \ b \ \text{in the subdomain} \ s
\]

- The frequency value of a bigram \( b \) is pre-computed and stored in a Match-Action Table (MAT).
- The lower the score, the more random the domain name.
- Example: the bigrams of “google” are: “$g”, “go”, “oo”, “og”, “gl”, “le”, “e$”

---

Proposed System

- Note that the system incorporates
  - Customized packet parsing and processing
  - Fine-grained measurements
  - Per-packet traffic inspection
  - Stateful memory processing at line rate
Evaluation

• Experimental setup
  ➢ Hundreds of GB of malware samples; 1,311 samples containing 50 DGA families\(^1\)
  ➢ We used samples that receive NXD responses containing domain names generated by DGAs\(^1\)
  ➢ The collected dataset was used to train ML models offline on a general-purpose CPU
  ➢ 80% of data was used for training and 20% for testing

---
The evaluation reports the accuracy (Acc), F1 score, and Precision (Prec) of different ML classifiers during the first eight NXD responses.

- The Random Forest (RF) model performed best.
- The Accuracy (Acc) starts at 92% from the first NXD response received and reaches 98% by the 8th NXD response.

<table>
<thead>
<tr>
<th>NXD count</th>
<th>RF Acc</th>
<th>F1</th>
<th>Prec</th>
<th>SVM Acc</th>
<th>F1</th>
<th>Prec</th>
<th>MLP Acc</th>
<th>F1</th>
<th>Prec</th>
<th>LR Acc</th>
<th>F1</th>
<th>Prec</th>
<th>GNB Acc</th>
<th>F1</th>
<th>Prec</th>
</tr>
</thead>
<tbody>
<tr>
<td>NXD 1</td>
<td>0.923</td>
<td>0.907</td>
<td>0.902</td>
<td>0.872</td>
<td>0.856</td>
<td>0.847</td>
<td>0.87</td>
<td>0.843</td>
<td>0.829</td>
<td>0.716</td>
<td>0.679</td>
<td>0.667</td>
<td>0.726</td>
<td>0.688</td>
<td>0.688</td>
</tr>
<tr>
<td>NXD 2</td>
<td>0.951</td>
<td>0.943</td>
<td>0.943</td>
<td>0.899</td>
<td>0.893</td>
<td>0.893</td>
<td>0.904</td>
<td>0.897</td>
<td>0.9</td>
<td>0.76</td>
<td>0.741</td>
<td>0.747</td>
<td>0.727</td>
<td>0.701</td>
<td>0.707</td>
</tr>
<tr>
<td>NXD 3</td>
<td>0.964</td>
<td>0.958</td>
<td>0.964</td>
<td>0.918</td>
<td>0.913</td>
<td>0.914</td>
<td>0.924</td>
<td>0.914</td>
<td>0.912</td>
<td>0.767</td>
<td>0.74</td>
<td>0.743</td>
<td>0.649</td>
<td>0.668</td>
<td>0.732</td>
</tr>
<tr>
<td>NXD 4</td>
<td>0.966</td>
<td>0.961</td>
<td>0.963</td>
<td>0.906</td>
<td>0.905</td>
<td>0.912</td>
<td>0.916</td>
<td>0.909</td>
<td>0.915</td>
<td>0.79</td>
<td>0.765</td>
<td>0.758</td>
<td>0.633</td>
<td>0.635</td>
<td>0.692</td>
</tr>
<tr>
<td>NXD 5</td>
<td>0.97</td>
<td>0.966</td>
<td>0.967</td>
<td>0.915</td>
<td>0.91</td>
<td>0.911</td>
<td>0.919</td>
<td>0.91</td>
<td>0.907</td>
<td>0.77</td>
<td>0.735</td>
<td>0.746</td>
<td>0.604</td>
<td>0.615</td>
<td>0.689</td>
</tr>
<tr>
<td>NXD 6</td>
<td>0.975</td>
<td>0.972</td>
<td>0.973</td>
<td>0.914</td>
<td>0.911</td>
<td>0.912</td>
<td>0.922</td>
<td>0.915</td>
<td>0.918</td>
<td>0.794</td>
<td>0.767</td>
<td>0.783</td>
<td>0.617</td>
<td>0.627</td>
<td>0.716</td>
</tr>
<tr>
<td>NXD 7</td>
<td>0.977</td>
<td>0.976</td>
<td>0.979</td>
<td>0.92</td>
<td>0.915</td>
<td>0.915</td>
<td>0.929</td>
<td>0.924</td>
<td>0.93</td>
<td>0.799</td>
<td>0.771</td>
<td>0.78</td>
<td>0.61</td>
<td>0.613</td>
<td>0.714</td>
</tr>
<tr>
<td>NXD 8</td>
<td>0.98</td>
<td>0.979</td>
<td>0.981</td>
<td>0.917</td>
<td>0.912</td>
<td>0.914</td>
<td>0.93</td>
<td>0.923</td>
<td>0.921</td>
<td>0.764</td>
<td>0.73</td>
<td>0.735</td>
<td>0.631</td>
<td>0.618</td>
<td>0.65</td>
</tr>
</tbody>
</table>

RF: Random Forest; SVM: Support Vector Machine; MLP: Multilayer perceptron; LR: Logistic Regression; GNB: Gaussian Naive Bayes
Evaluation

• The scheme can accurately characterize traffic flows (context-aware features)
• Interarrival times between NXDs of DGA families with the largest number of samples
Evaluation

- Comparison of the feature extraction time of the proposed approach vs EXPLAIN$^1$
  - The proposed approach runs on the switch data plane
  - EXPLAIN runs on a general-purposed CPU with 64 GB RAM, 2.9 GHz processor with eight cores

---

Conclusion

• This presentation briefly described the evolution of networking devices, from legacy (monolithic) units to SDN to P4 PDP switches

• It discussed the capabilities offered by PDP switches to enable end programmers to produce fine-grained measurements, customized parsers and functions, and line-rate computation

• Such capabilities were applied to solve two different problems
  - Buffer sizing problem, where programmability was enabled in non-programmable devices, to solve the buffer sizing problem via a passive deployment of P4 switches
  - DGA problem, where the P4 application was able to detect and classify DGAs using a combination of DNS deep packet inspection and traffic characterization
Conclusion

- The previous two are only a couple of examples of the impressive work produced by the P4 community, which suggests that deep programmability (switches, smart NICs, etc.) will continue in the near future.

---

Contact Information

Jorge Crichigno
College of Engineering and Computing, University of South Carolina
jcrichigno@cec.sc.edu
http://ce.sc.edu/cyberinfra