USENIX nsdi '26

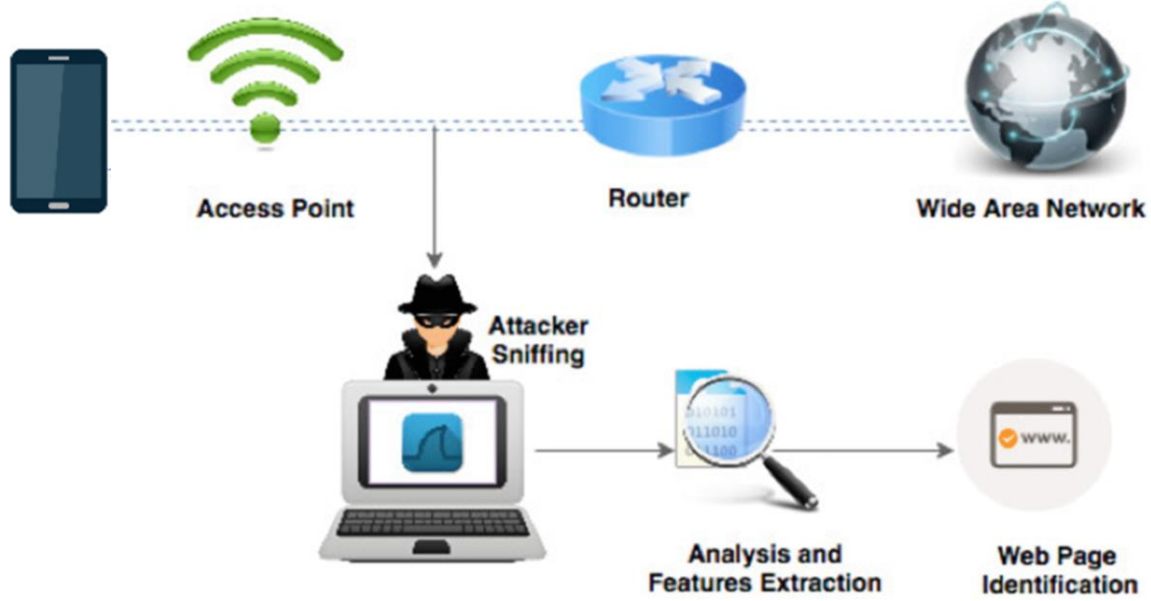
Defending against Traffic Analysis Attacks with Flexible In-Network Obfuscation

Guorui Xie · Qing Li · Zhenning Shi · Gianni Antichi · Yijia Zhu
Kejun Li · Changxing Weng · Sebastiano Miano · Yong Jiang · Mingwei Xu

Pengcheng Laboratory
Tsinghua University
POLITECNICO MILANO 1863
XIDIAN UNIVERSITY

Traffic Analysis Attacks

The attacker uses *side-channel information*, e.g., sizes/directions/timing/IPs of encrypted packets to predict user activities by DNN



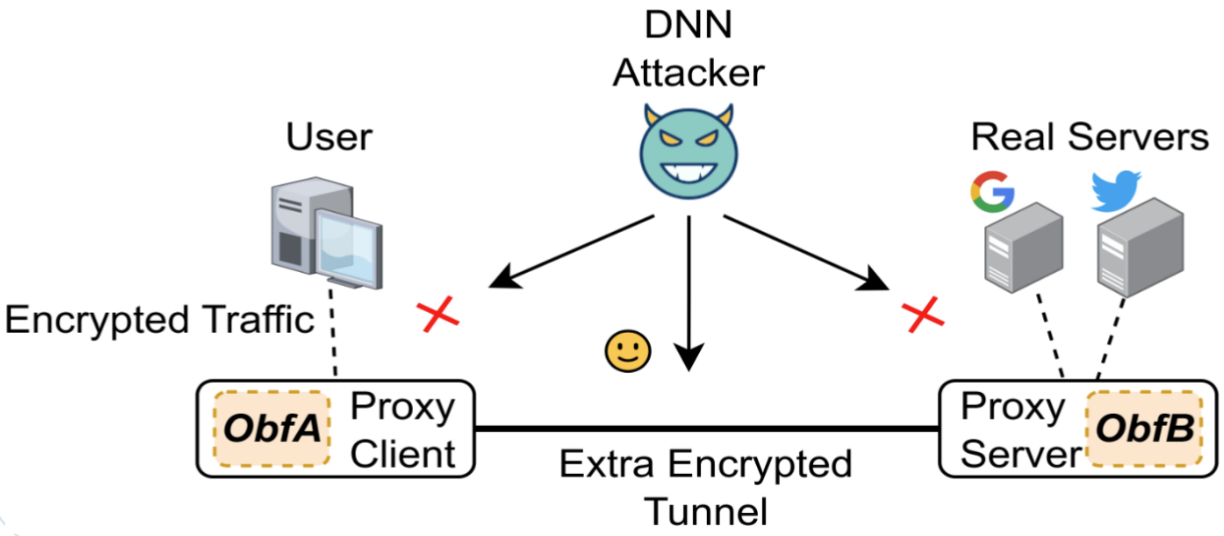
Web fingerprinting example

Malicious examples:

1. When and which website you are most interested => *targeted phishing*
2. What IoT devices are in your house => hijacking specific camera for *surveillance*

Exampled Defense Explanation

Multiple defenses like BLANKET/WTF-Pad/Ditto use a similar framework



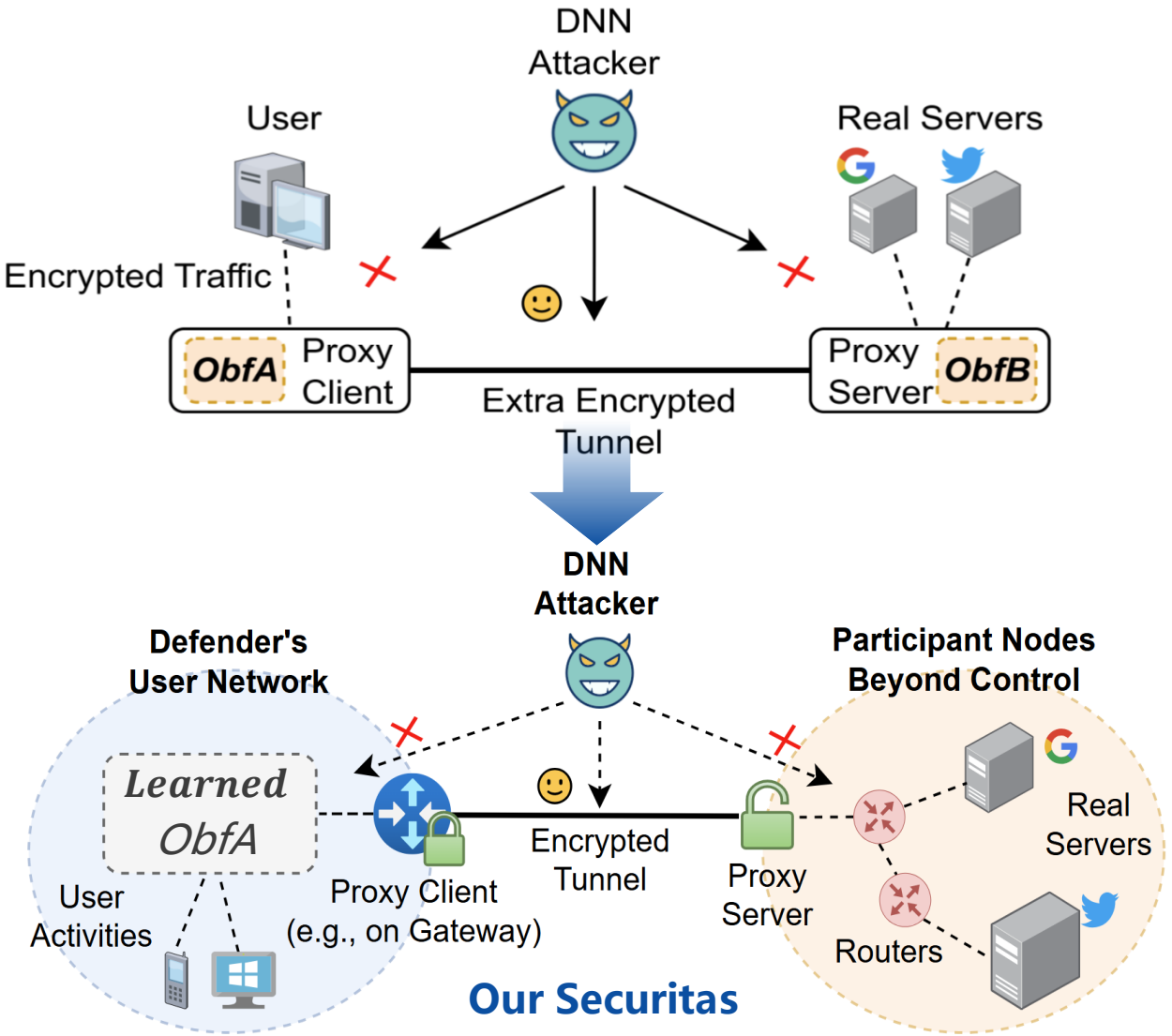
Key points:

1. A proxy tunnel is built for:
   - Hiding obfuscation details
   - Covering the real visited IPs
2. Obfuscators work in pairs: e.g., ObfA pads packets, and ObfB removes paddings inversely

✗ Inflexible as the proxy server (e.g., NordVPN) is beyond our control

✗ Padding packets with an unoptimized strategy is costly (Ditto may yield 140+% bandwidth overhead)

Our Redesign



This redesign meets the properties:

- ✓ **Multi-Info Obfuscation**: ObfA changes directions , sizes, timing; proxy hides IPs
- ✓ **Low Bandwidth Overhead**: *In-network* fast obfuscation by learned strategies
- ✓ **Flexible Obfuscation**: No ObfB on uncontrolled Internet nodes

Three Main Challenges

1. Without ObfB, how could we de-obfuscate the outbound traffic to the server, and obfuscate the inbound traffic?
   - ➢ *Obfuscation based on the IP behaviors of fragments and TTLs*
2. Given the unknown attack DNN and non-differentiable obfuscation operators, how could we train the obfuscation strategy?
   - ➢ *The policy gradient derived from the proxy DNN and loss penalty*
3. How to implement in-network ObfA for different programmable devices (Tofino/FPGA/Software etc.)?
   - ➢ *Unify obfuscation operations in a clone-then-modify manner*

Obfuscation Operation 1: Packet Fragmentation

Key observation: Routers fragment packets for forwarding when the next-hop MTU is smaller

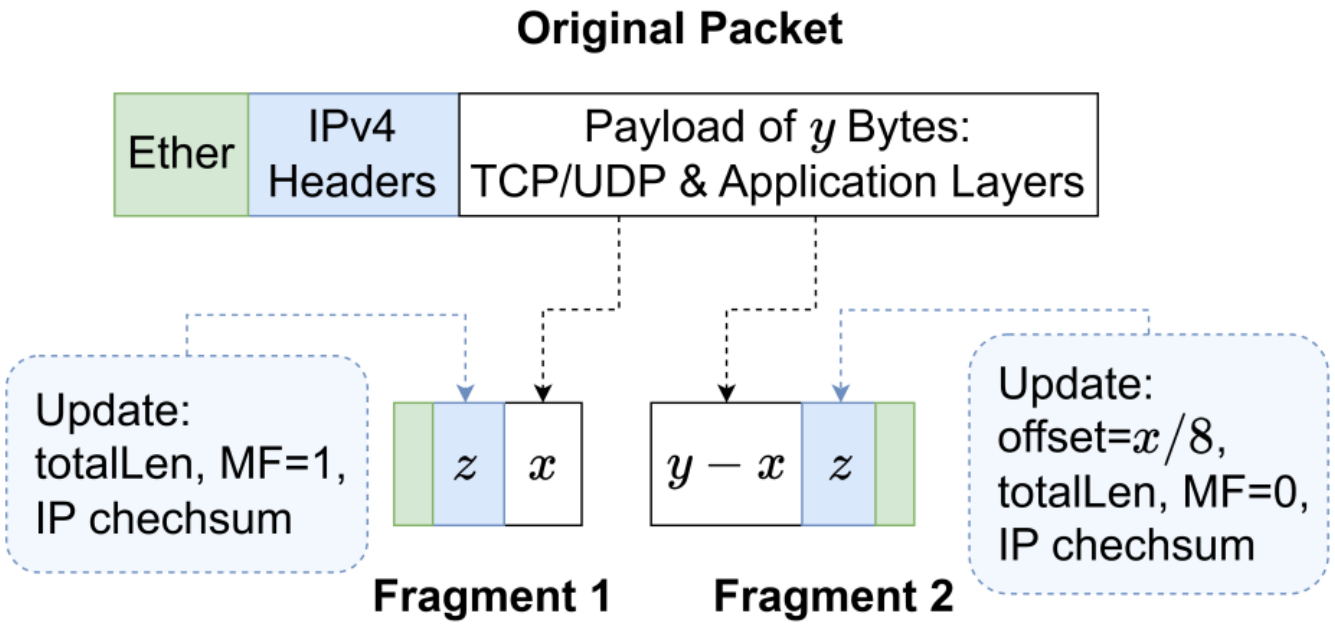


Illustration of the fragmentation with IP header modification

- a_i in strategy $\boldsymbol{a}^* = \{a_1 \dots a_n\}$ decides the $outbound$ packet p_i with l bytes to be two pieces of sizes a_i and $l - a_i$
- By adding fragments as new packets, the observed size/direction/timing features are changed

But only covers the outbound traffic…

Obfuscation Operation 2: Small-TTL Packet Insertion

Key observation: When TTL is reduced to zero, routers will discard the packet and probabilistically send back a ICMP reply

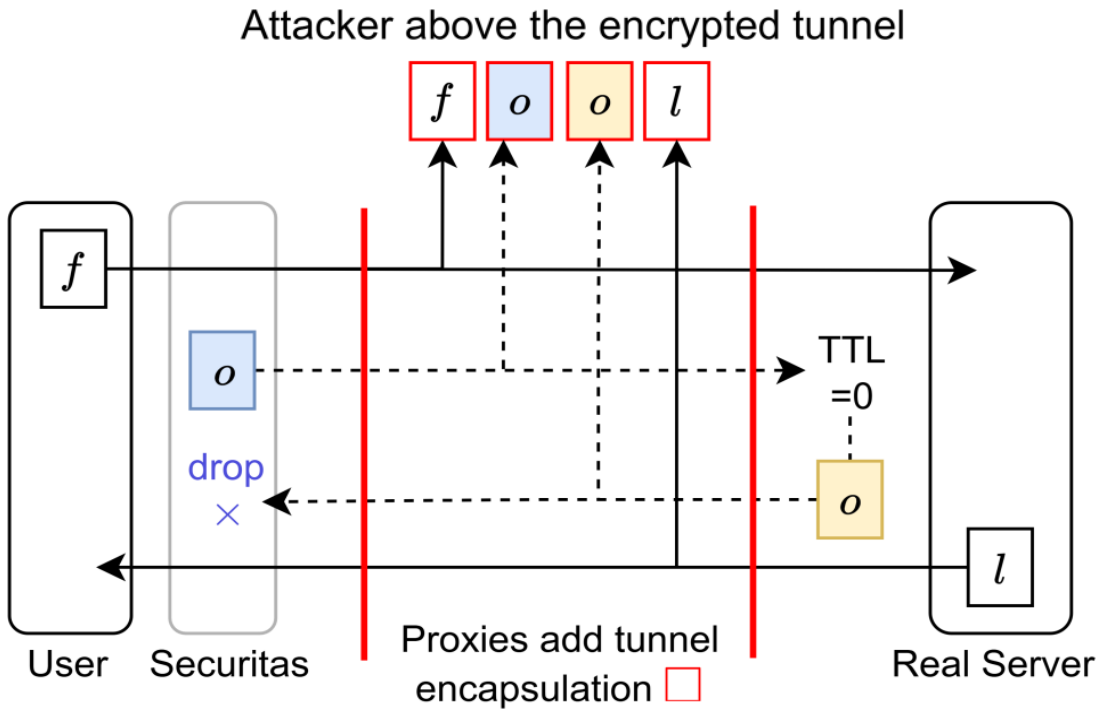


Illustration of using small-TTL packets for protecting inbound traffic

- a_i in $\boldsymbol{a}^*$ alternatively decides the inserted $outbound$ packet of a_i bytes after p_i
- The encrypted tunnel makes both inserted packets and returned ICMP packets stealthy

But inserting new packets introduces more extra bytes…

Learning the Trade-off

1 The Neural Agent

$$out, h = \mathrm{LSTM}(a_{i-1}, h)$$
$$P(a_i | a_1, \ldots, a_{i-1}) = \mathrm{softmax}(out)$$

$$\hat{\boldsymbol{p}} = \Phi(\boldsymbol{p}, \boldsymbol{a})$$

2 Loss Function

$$\ell = \mathrm{Confidence}(f(\hat{\boldsymbol{p}})) + \mathrm{ByteRatio}(\boldsymbol{p}, \hat{\boldsymbol{p}}) + \mathrm{Similar}(\boldsymbol{p}, \hat{\boldsymbol{p}})$$

The probability that the proxy attack DNN predicts obfuscated $\hat{\boldsymbol{p}}$ correctly

$$\frac{\hat{\boldsymbol{p}}.bytes - \boldsymbol{p}.bytes}{\boldsymbol{p}.bytes}$$

New bytes introduced from $\boldsymbol{p}$ to $\hat{\boldsymbol{p}}$

As the real attacker is unknown, we should reduce the similarity between $\boldsymbol{p}$ and $\hat{\boldsymbol{p}}$

3 Policy Gradient to Minimize ℓ

$$\theta = \theta - \lambda \frac{1}{B} \sum_{i=1}^{B} \ell_{\Phi(\boldsymbol{p}_i, \boldsymbol{a}_i)} \cdot \nabla_\theta \log(P_{\boldsymbol{a}_i})$$

Deployment with P4

Algorithm 1 Obfuscation Function $\Phi(\boldsymbol{p}, \boldsymbol{a})$

Input: Defender's preference $Pr \in (0, 1.0)$.
Output:Return processed $\boldsymbol{p}$ as $\hat{\boldsymbol{p}}$.

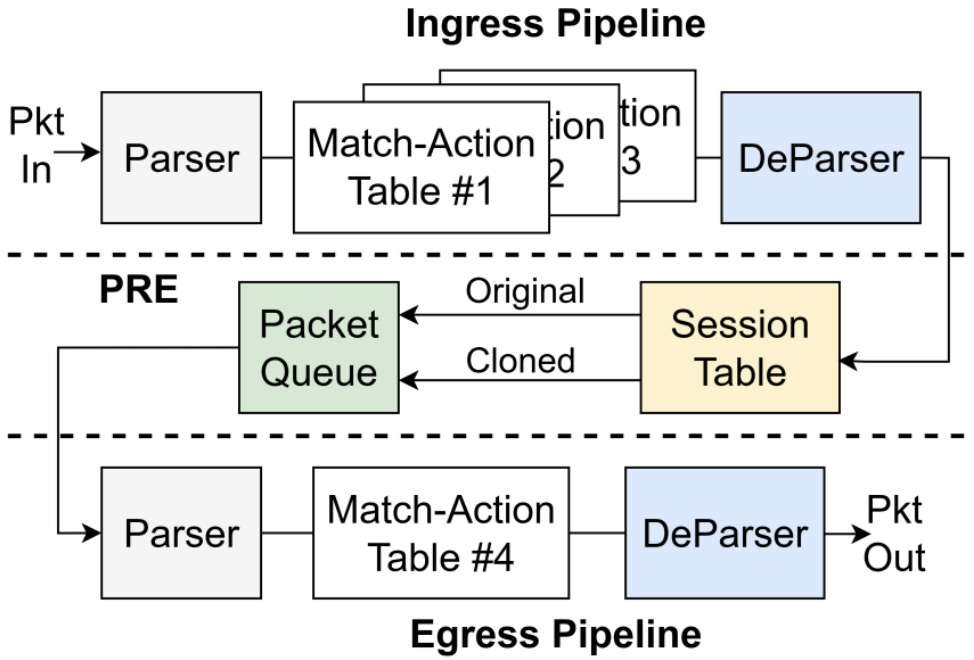
```
1:  Random.seed = 0.
2:  for a_i ∈ a do
3:      if a_i is 0 or p_i is inbound then
4:          Continue.
5:      end if
6:      rnd = Random().
7:      if rnd < Pr then
8:          Replace p_i by splitting it into two fragments of a_i
            and p_i.size − a_i bytes.
9:      else
10:         Insert a fake packet with a small TTL and a_i bytes
            immediately after p_i.
11:     end if
12: end for
13: if training then
14:     Insert ICMP packets somewhere after the small-TTL
        packets according to §4.2.
15: end if
```

$a_i = 0$ enables variable-length and non-contiguous obfuscation

Fragment

Small-TTL packet

Clone-then-Modify on the Portable Switch Architecture of P4



Ingress Pipeline: count packet i, match entry a_i for fragment/insertion

PRE (TM in Tofino): clones and truncates packets

Egress Pipeline: modify headers accordingly

Evaluation Settings

We conduct thorough experiments, comparing Securitas with SOTA schemes on datasets, synthetic traffic, and real-world tests

Dataset Test

- 3 attack scenarios: website, IoT, and App fingerprinting
- 5 SOTA defend against 9 attack DNNs
- Models are trained by PyTorch 1.9.1 on 2080Ti GPUs

Synthetic Traffic Test

- Hardware testbed of Tofino and FPGA
- 20 to100Gbps traffic generated by Cisco TRex 3.04
- User experience tested by iPerf 2.1.5, pywb 2.8.3

Real-World Test

- Visit 26 web servers and 2 IoT hubs through NordVPN
- Hardware (Tofino) and Software (BMv2, eBPF) testbed
- User concurrency testing by Locust 2.25

Dataset Test Results

	Website Fingerprinting			IoT Fingerprinting			Application Classification		
	DF	AWF	Var-CNN	ANN	LSTM	BILSTM	App-Net	FS-Net	Transformer
No Defense	98.78%	98.65%	99.40%	90.20%	83.30%	86.53%	78.80%	79.80%	72.60%
Minipatch	8.13%	10.59%	8.94%	47.96%	40.12%	41.76%	73.49%	67.43%	38.62%
BLANKET	14.42%	17.14%	9.84%	44.50%	48.72%	50.85%	19.20%	**16.20%**	24.60%
WTF-PAD	18.98%	9.89%	14.01%	29.85%	32.62%	32.30%	**16.66%**	16.66%	20.95%
Ditto	3.77%	2.36%	3.71%	25.15%	23.40%	28.76%	22.40%	22.80%	**8.20%**
Securitas	3.01%	1.96%	3.51%	23.49%	25.62%	24.27%	31.20%	37.80%	16.8%
Securitas*	**3.01%**	**0.75%**	**2.71%**	**23.49%**	**22.58%**	**20.07%**	31.20%	34.00%	14.40%

Attack Evasion

The percentage of flows where the attacker predict classes correctly

Securitas reduces accuracy of Var-CNN by **96+%**

	Website Fingerprinting			IoT Fingerprinting			Application Classification		
	DF	AWF	Var-CNN	ANN	LSTM	BILSTM	App-Net	FS-Net	Transformer
Minipatch	3.39%	2.98%	2.63%	21.70%	41.77%	43.12%	129.36%	310.34%	279.20%
BLANKET	26.45%	23.35%	26.61%	25.25%	25.14%	25.13%	21.92%	21.90%	21.90%
WTF-PAD	29.09%	29.09%	29.09%	133.15%	133.15%	133.15%	257.33%	257.33%	257.33%
Ditto	92.13%	92.13%	92.13%	186.57%	186.57%	186.57%	265.11%	265.11%	265.11%
Securitas	3.31%	3.31%	3.31%	4.15%	4.15%	4.15%	6.21%	6.21%	6.21%
Securitas*	**3.31%**	**3.30%**	**3.29%**	**4.15%**	**4.06%**	**4.08%**	**6.21%**	**6.11%**	**6.06%**

Bandwidth Overhead

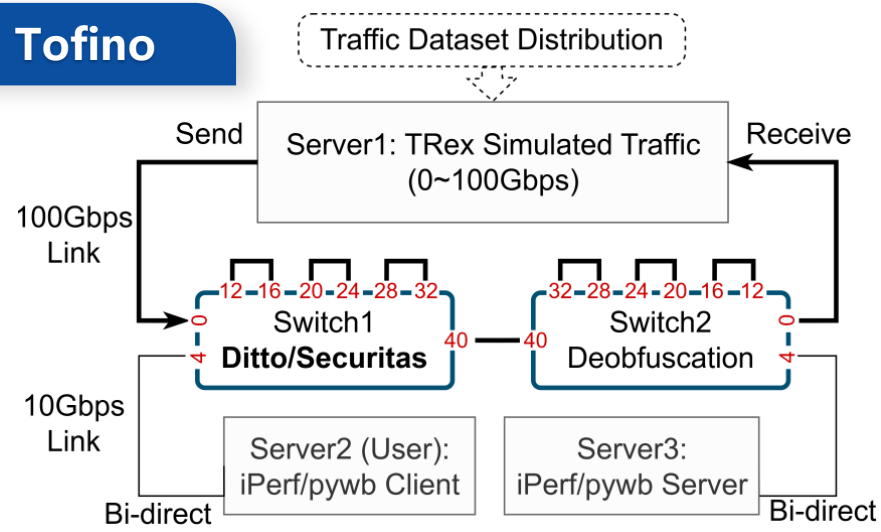
The percentage of extra bandwidth consumed

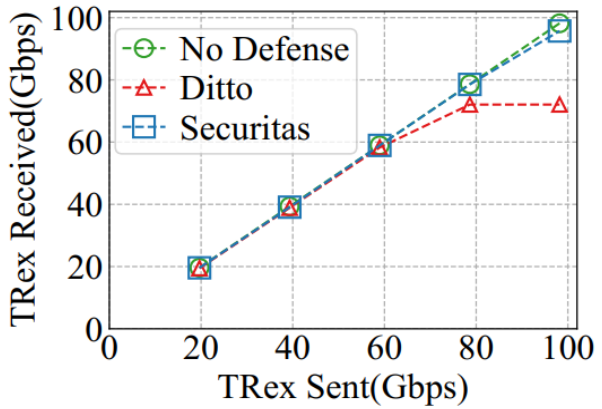
Securitas is **42.69×** lower compared to Ditto

- Securitas is the case where we know the exact attack DNN during the agent training.*

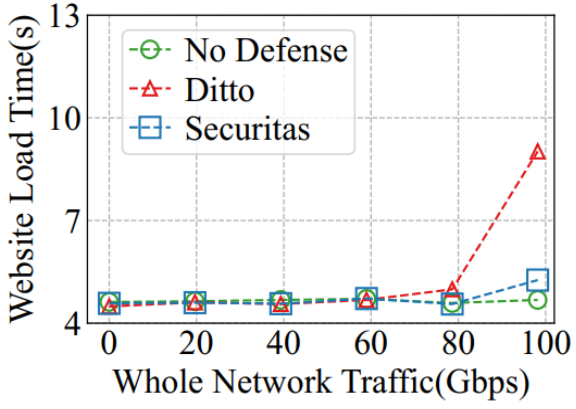
Synthetic Traffic Test Results

Tofino



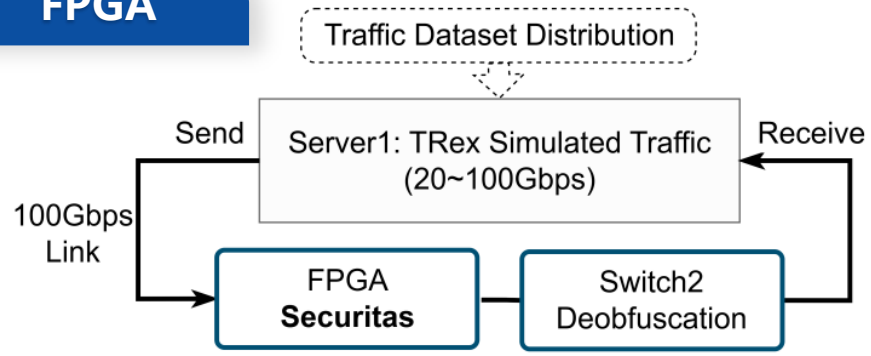


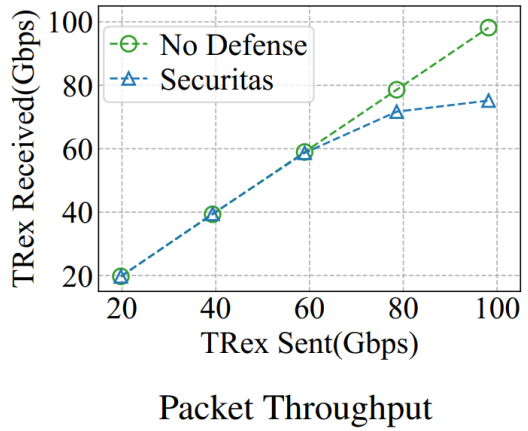
TRex sent vs. received



Single user's iQIYI load time

FPGA





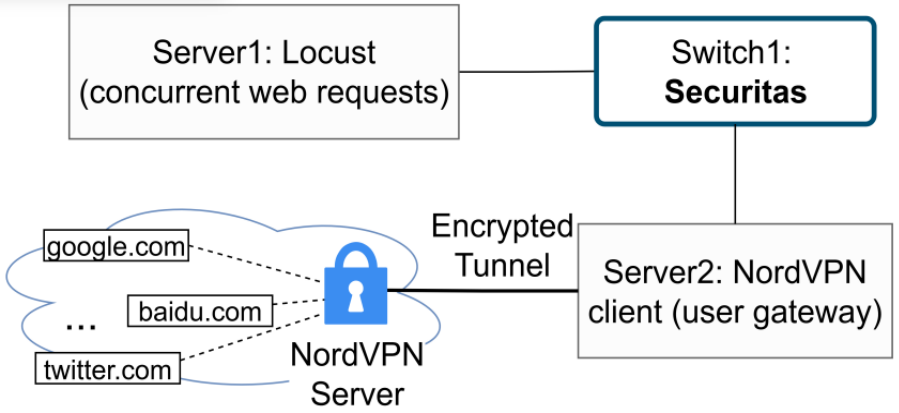
Packet Throughput

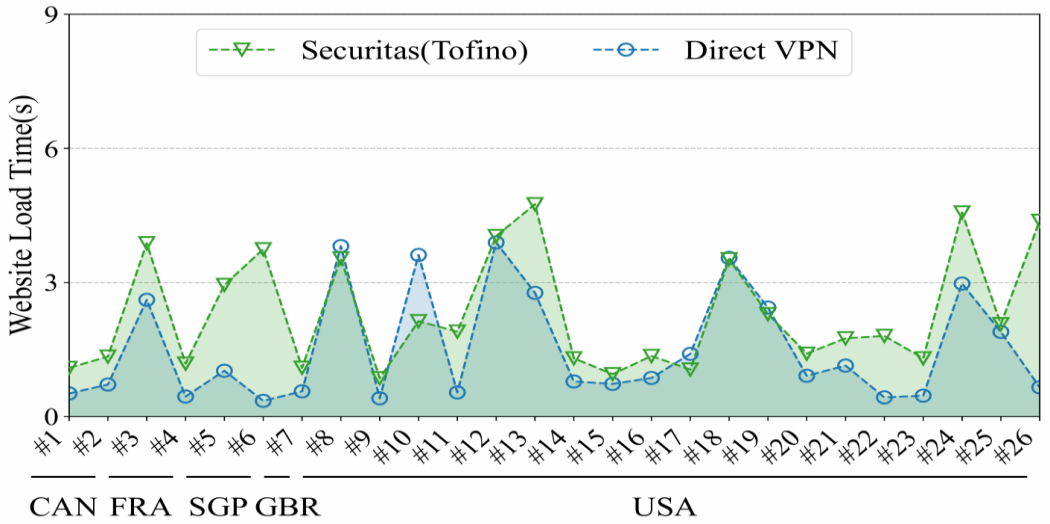
Analysis

- 95Gbps vs. 72Gbps (Ditto) on Tofino
- 75Gbps on FPGA
- iQIYI Web load time is1.8x faster than Ditto

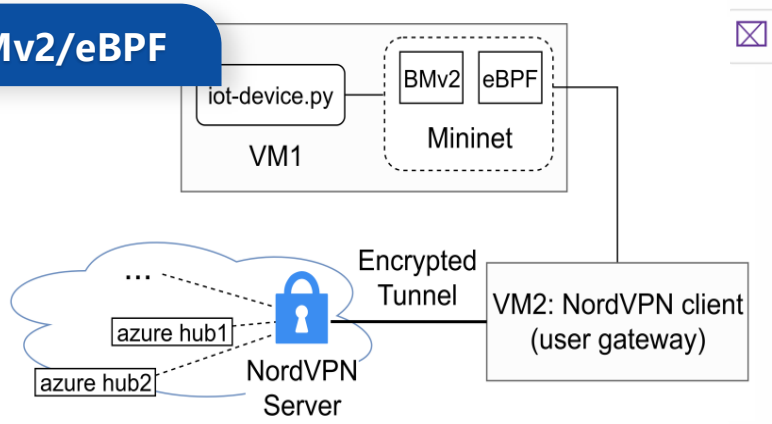
Real-World Test Results

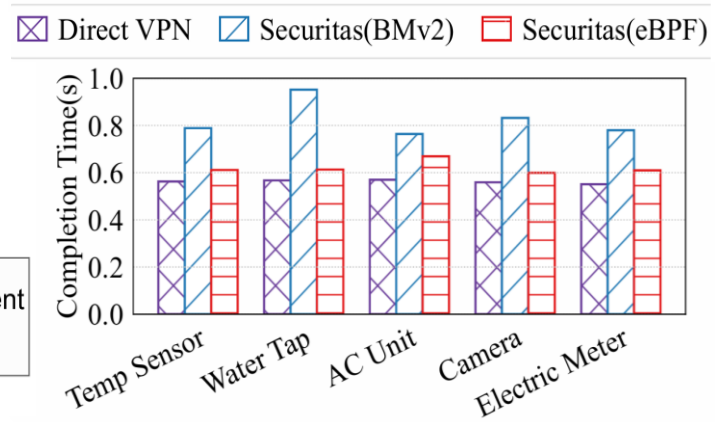
Tofino





BMv2/eBPF





Tofino Test ~70% websites has negligible delay (≤0.15s)

BMv2 eBPF IoT devices work with an extra latency of 0.2-0.3s

Conclusion

Securitas is an in-network obfuscator for flexible and low-overhead defense:

1. Based on packet fragmentation, small-TTL packet insertion, and implicit ICMP replies, Securitas provides flexible traffic obfuscation
2. With the learning optimization, Securitas effectively balances attack evasion and bandwidth overhead
3. Our dedicated P4 implementation makes Securitas run on hardware and software
4. Dataset and real-world tests confirm our superiority, e.g., reducing attack accuracy by 96% and having a web load delay ≤0.15s.

Thank you! Email me: xiegrr@gmail.com and check code: https://github.com/xgr19/Securitas