Tolerating Overload Attacks Against Packet Capturing Systems

Antonis Papadogiannakis — *FORTH-ICS, Greece*

Michalis Polychronakis — *Columbia University, USA*

Evangelos P. Markatos — *FORTH-ICS, Greece*



USENIX ATC 2012



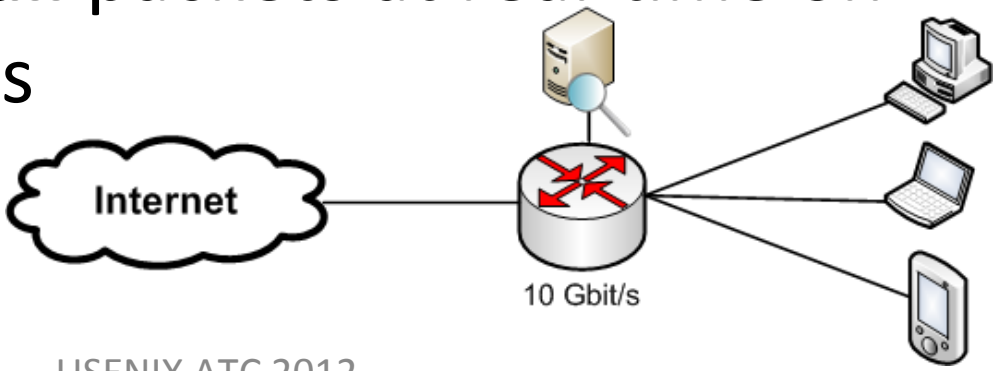
Network Traffic Monitoring Applications

- Network-level intrusion detection
- Network traffic classification
- Next-generation firewalls



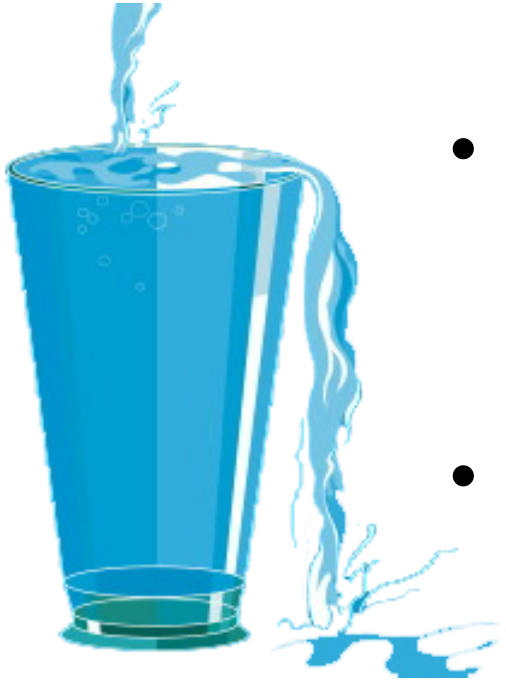


- Based on packet capturing systems
- Need to analyze **all** packets at real-time on multi-Gigabit links



What Happens in Case of Overload?

- Short-term traffic or processing bursts
- Handled with **limited buffering** in main memory



- When memory buffer becomes full, excess packets are ***randomly*** dropped
- **Important** packets may be lost
  - e.g. missed attacks

Even Worse: Overload Attacks



- Algorithmic complexity attacks
  - Exploit the differences between **average case** and **worst case** complexity
  - **Low bandwidth** attacks
  - Each packet results to orders of magnitude slower processing
- Traffic overload attack
  - Higher rates than the system can handle

Existing Solutions

- Provisioning
  - Tune the system for worst case scenarios
- Thresholds
- Algorithmic solutions
  - Reduce the difference between average and worst case performance
- Selective discarding
  - Discard the less important packets under overload

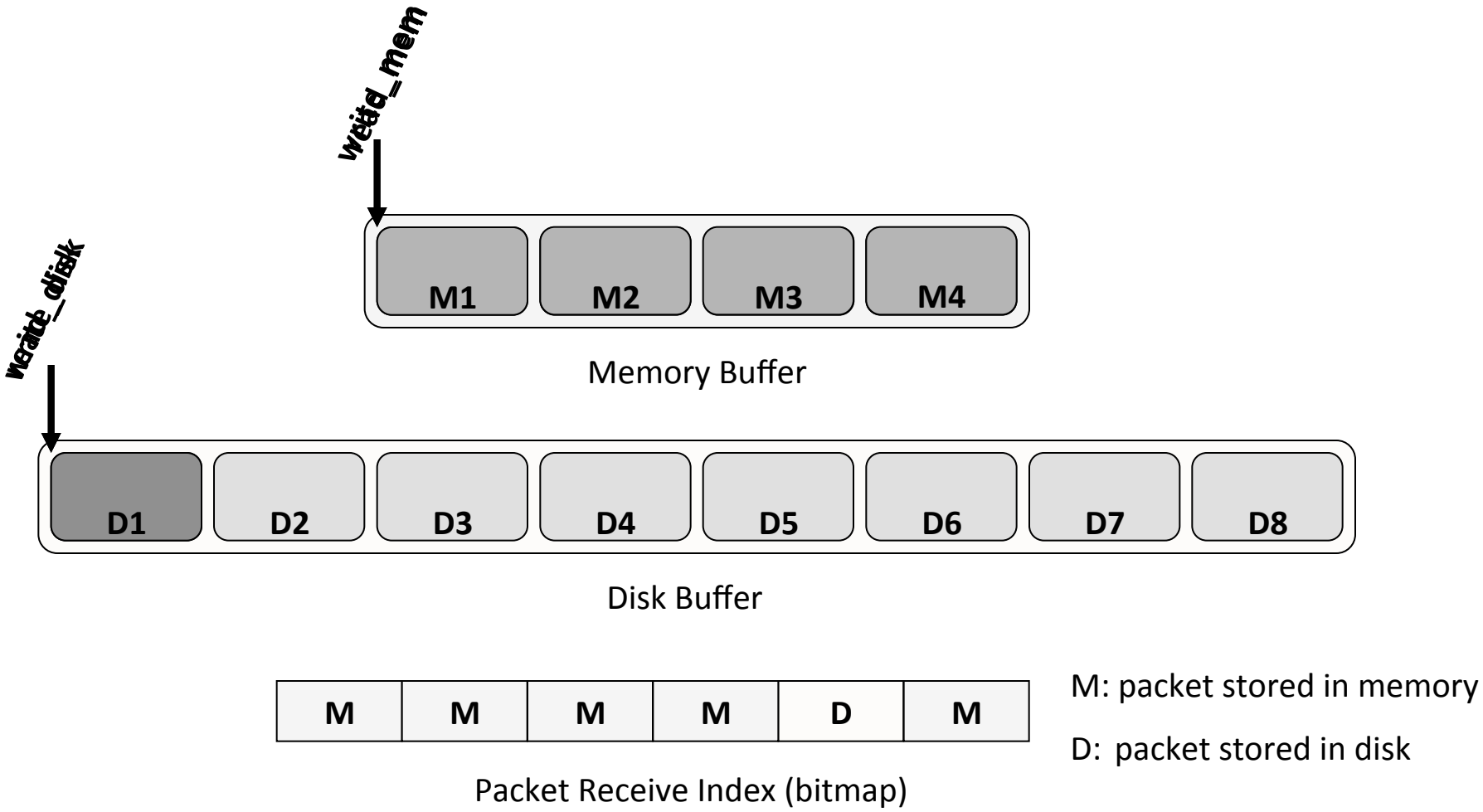
Our Approach: *Selective Packet Paging*



- Two layer memory management
  - Excess packets are buffered to **secondary storage**
  - **All** packets are analyzed

- Detection with randomized timeout interval
  - Detects malicious packets that **slowdown** the system
  - Send **these** packets to secondary storage

Packet Paging: Two-layer Memory Management



Packet Paging: Two-layer Memory Management



- Modern disks
  - Three orders of magnitude **more storage** than main memory
  - Disk **throughput** increases, multiple disks organized in RAID0
- *Packet capturing and storing thread* has higher priority than *packet processing thread*
- *Disk I/O thread* prefetches packets from disk and optimizes throughput

Selective Packet Paging

- With packet paging, both benign and malicious packets are buffered to disk
  - Increased delay for benign packets

- With ***selective*** *packet paging* only **malicious** packets are buffered to disk
  - Benign packets do not experience any delay
  - Malicious packets experience almost the same delay

Malicious Packet Detection

- Malicious packets: packets with unusually high processing times
- The problem: detecting malicious packets that slowdown the system with minimum overhead

1st way: Using the Timestamp Counter

```
read_timestamp();
process_packet();
read_timestamp();
```



User level, not interrupt driven

Problem1: the packet cannot be evicted

Problem2: measures the total system time, not application's CPU time

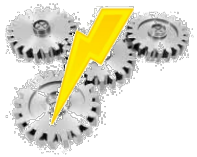
2nd way: Setting a Timer per-packet

```
set_timer(interval);

process_packet();
```







Kernel based, interrupt driven, timer expiration based on application's CPU time

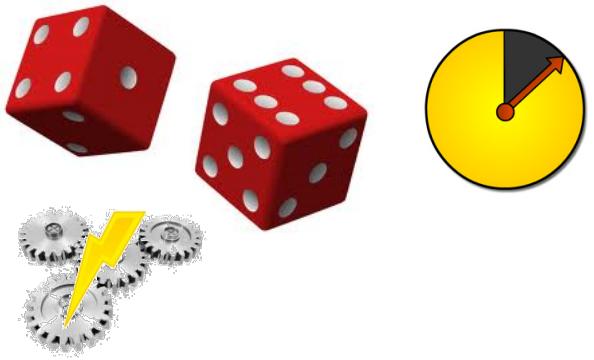
Problem 1: runtime overhead for setting a timer per packet

Problem 2: a constant or predictable interval can be used by attackers to evade detection

Randomized Timeout Interval

```
set_timer(random interval);

process_packets();
```



At timer expiration: check if the same packet is being processed

The timeout interval is chosen **randomly** from [low, high]

Detection Probability

$$P(\text{det}) = \frac{a \times d}{d + t} \times \frac{(d - low)^2}{2 \times d \times (high - low)}$$

a is the percentage of attack packets
d is the average processing time of attack packets
t is the average processing time of benign packets

Probability of detecting an attack after *N* timeouts:

$$1 - (1 - P(\text{det}))^N$$

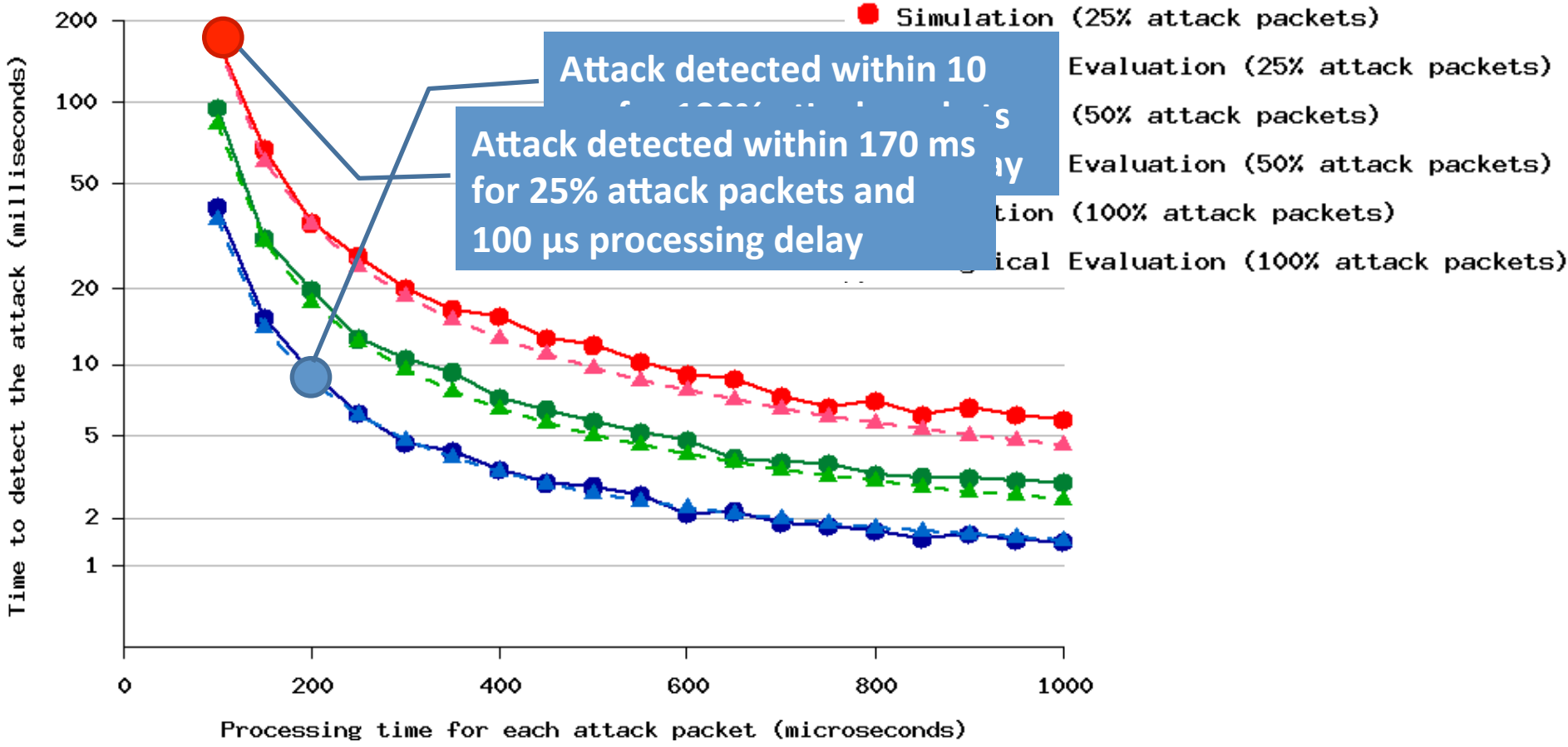
approaches 1 very fast

Average detection time:

$$T = 1 / P(\text{det}) + 1 \text{ timeouts}$$

$$T \times (high - low) / 2 \text{ μs}$$

Detection Time



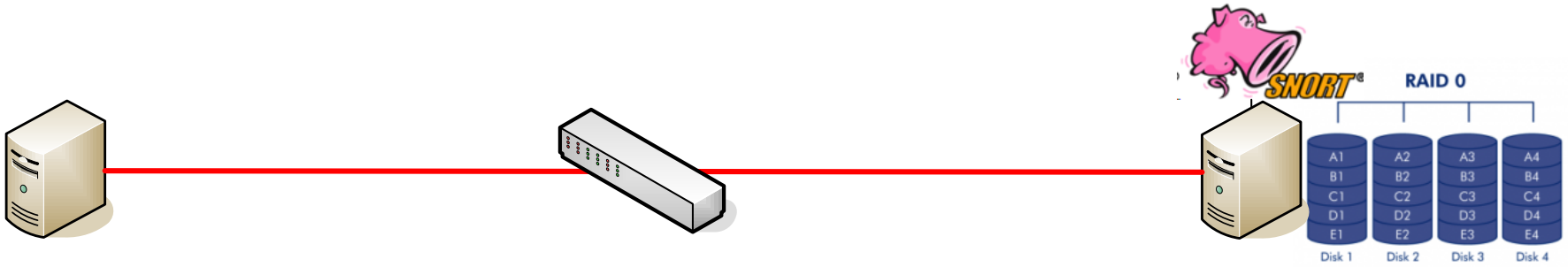
low=50 µs, *high*=1000 µs

Implementation

- Within the popular libpcap packet capture library

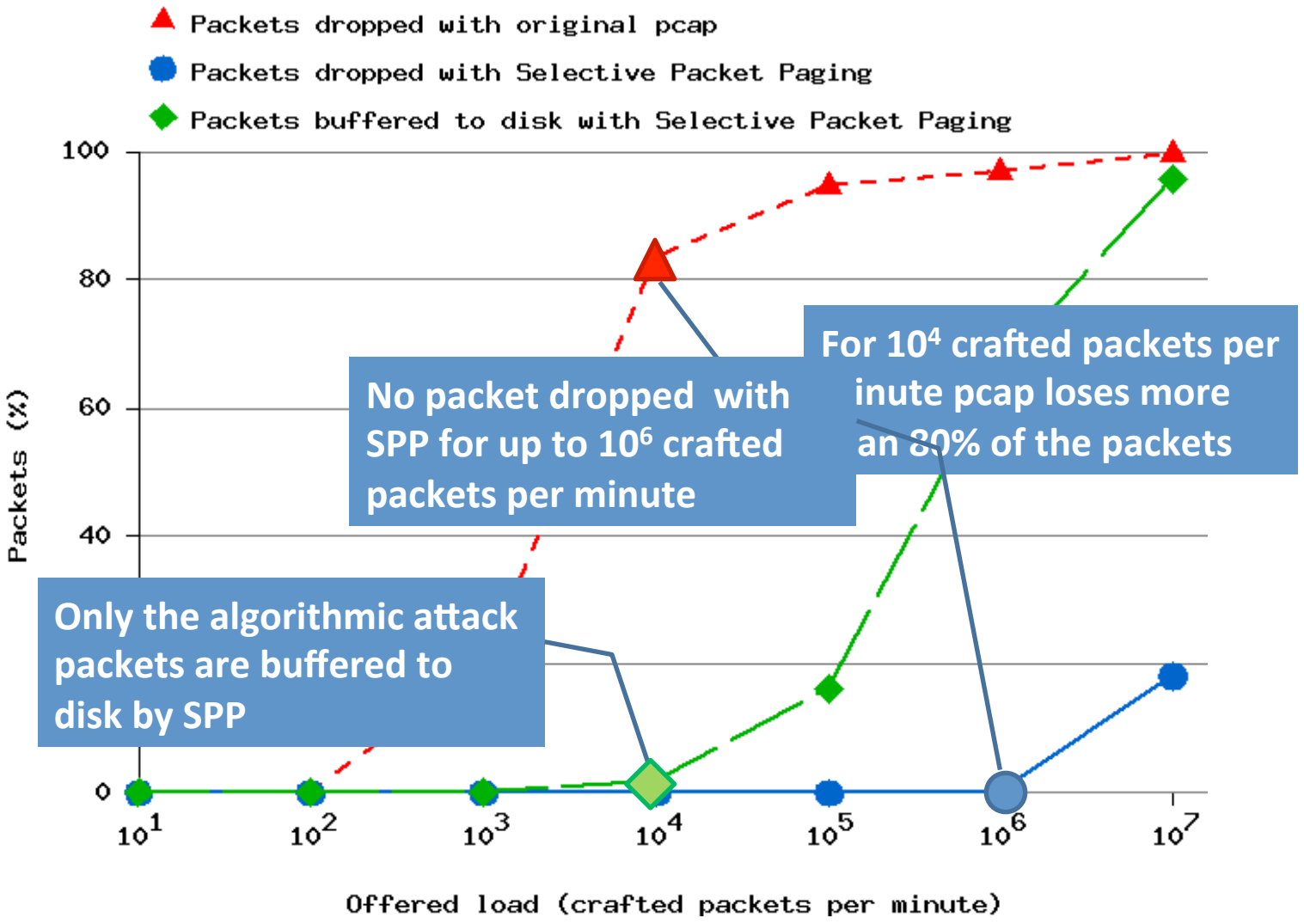
- Existing applications can use selective packet paging **without** code modifications

Experimental Evaluation

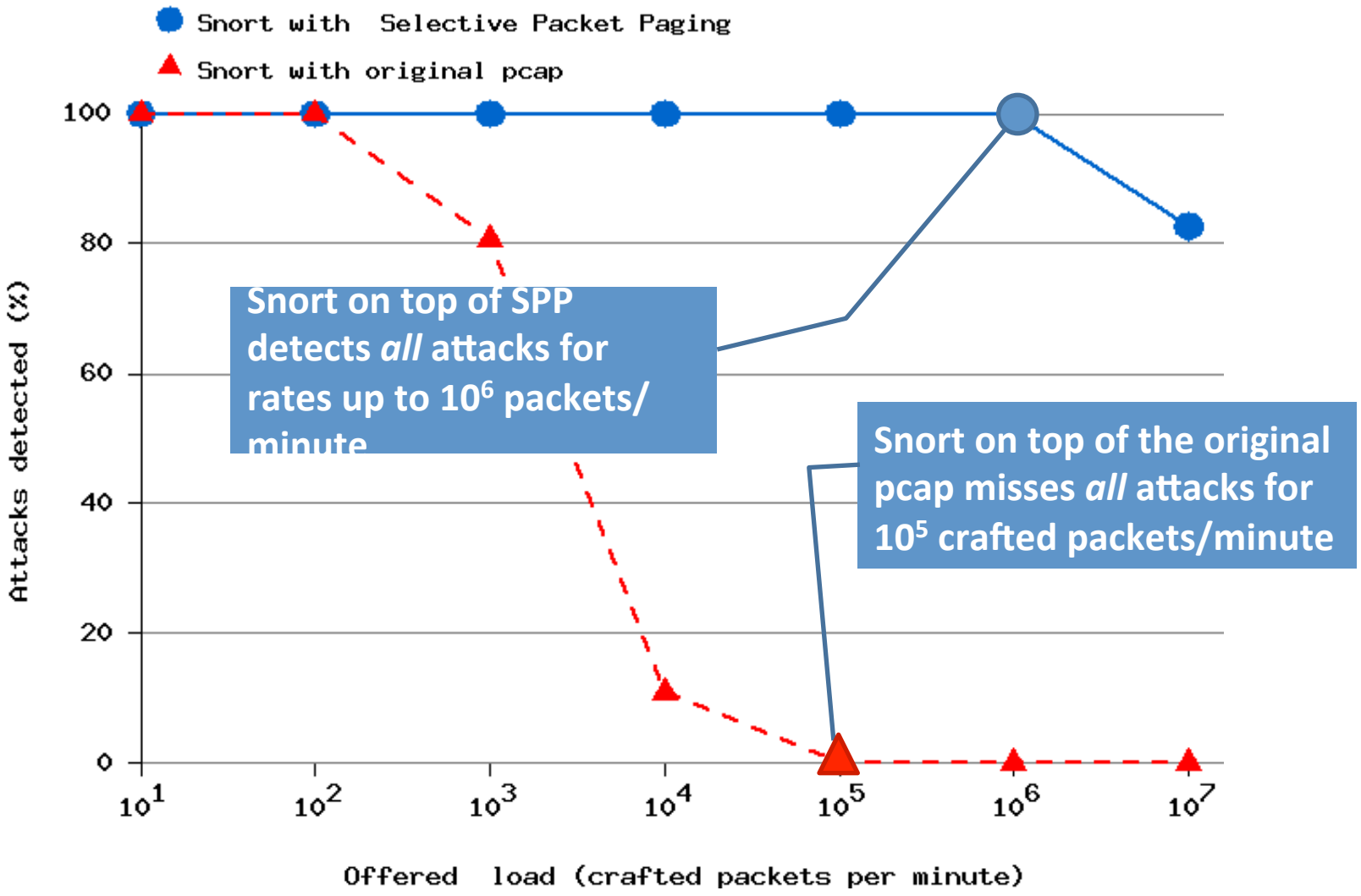


- Two experiments
  - **Algorithmic complexity** attack against Snort's regular expression matching using excessive backtracks
  - **Traffic overload** attack against Snort

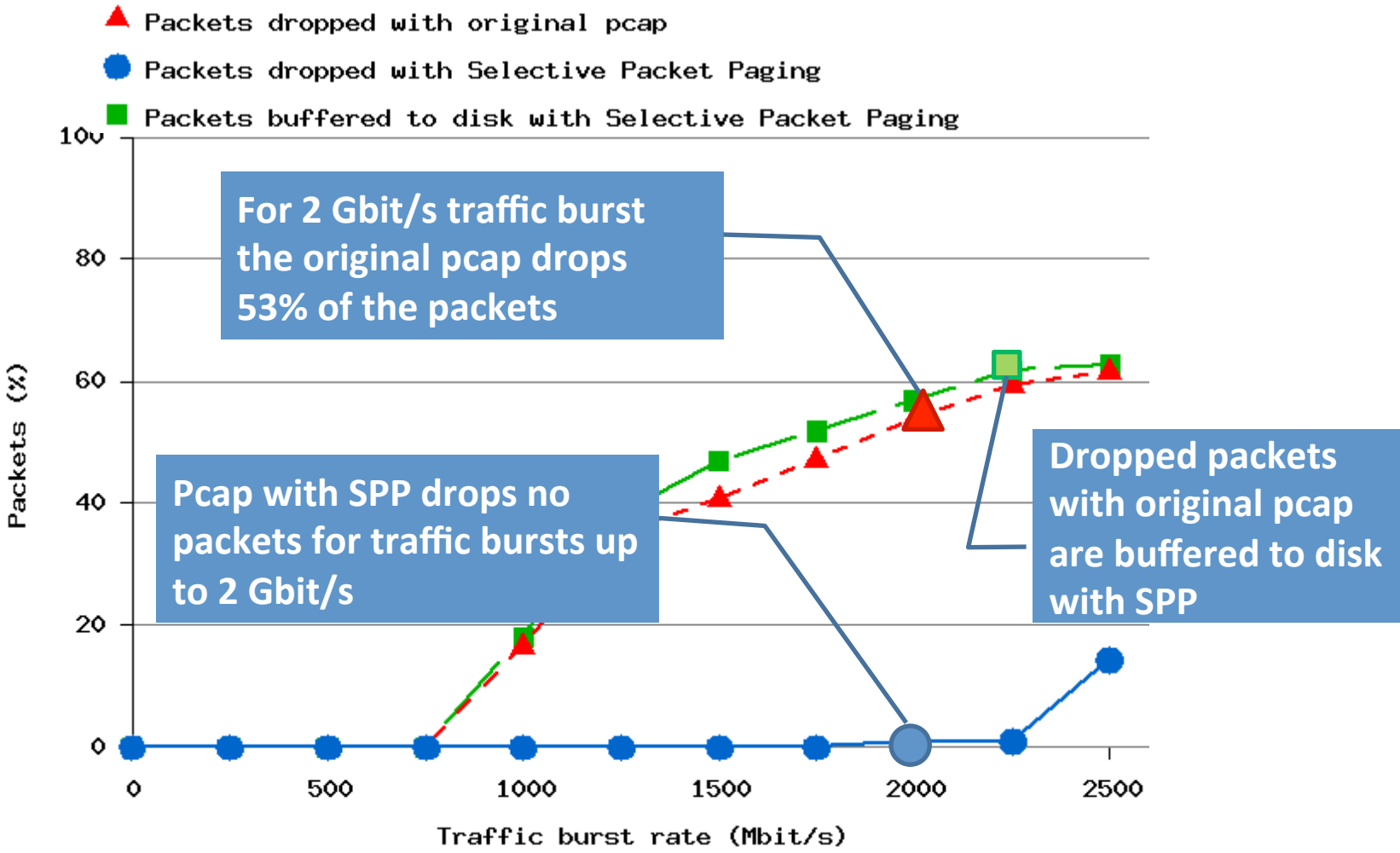
Algorithmic Complexity Attack: Dropped Packets



Algorithmic Complexity Attack: Detected Attacks

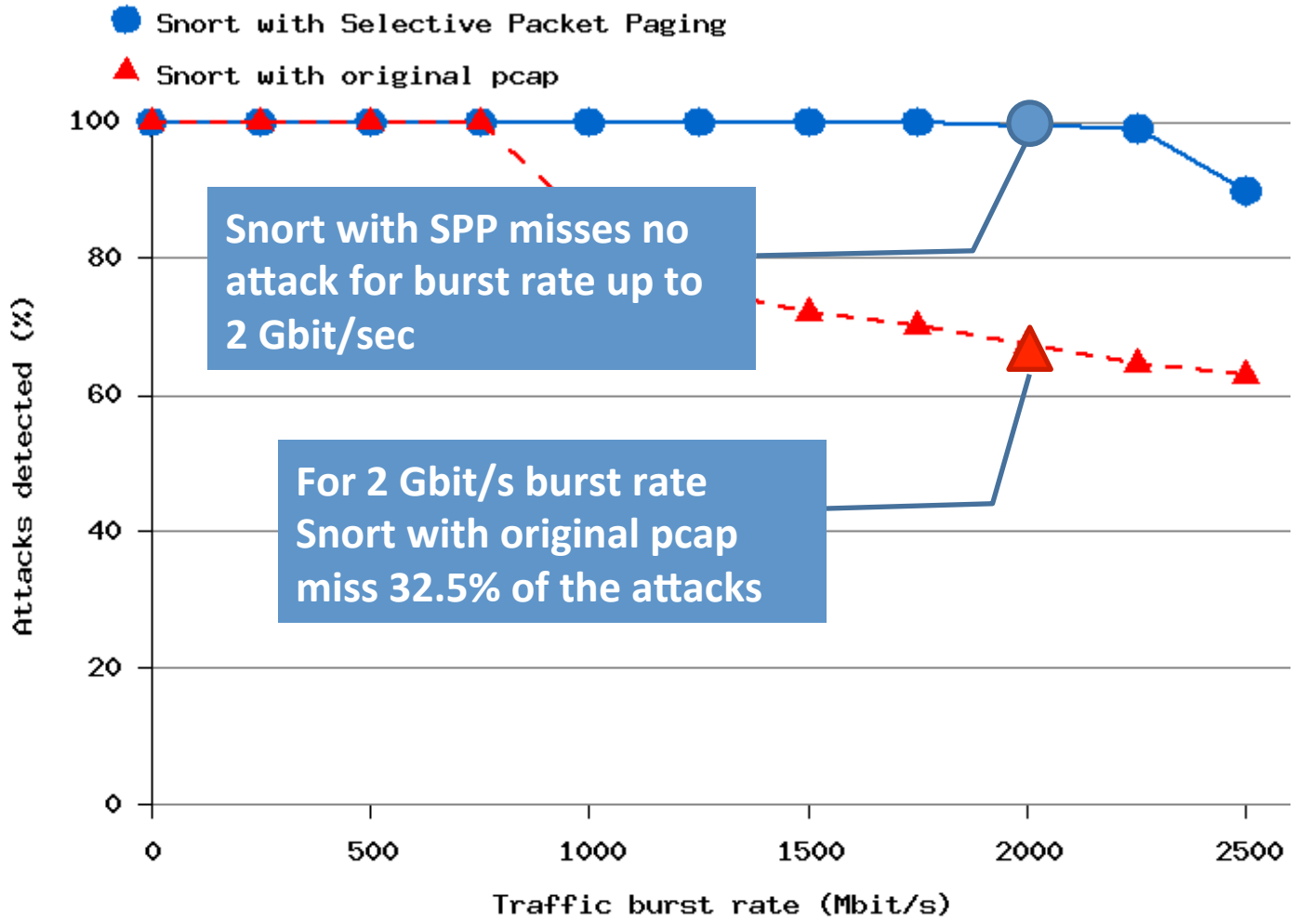


Traffic Overload Attack: Dropped Packets



30-seconds long traffic bursts

Traffic Overload Attack: Detected Attacks



30-seconds long traffic bursts

Summary

- Difficulty to predict and avoid **overloads**
- As monitoring applications become more complex, they may be more vulnerable to **algorithmic attacks**
- *Selective Packet Paging*: **generic** defense against **unknown** algorithmic attacks and **unpredictable** overloads
  - Two-layer memory management, using secondary storage
  - Detection based on randomized timers

- Disk throughput increase as network speeds increase
  - Disks can be organized in RAID0

Tolerating Overload Attacks Against Packet Capturing Systems

Antonis Papadogiannakis papadog@ics.forth.gr
Michalis Polychronakis mikepo@cs.columbia.edu
Evangelos P. Markatos markatos@ics.forth.gr