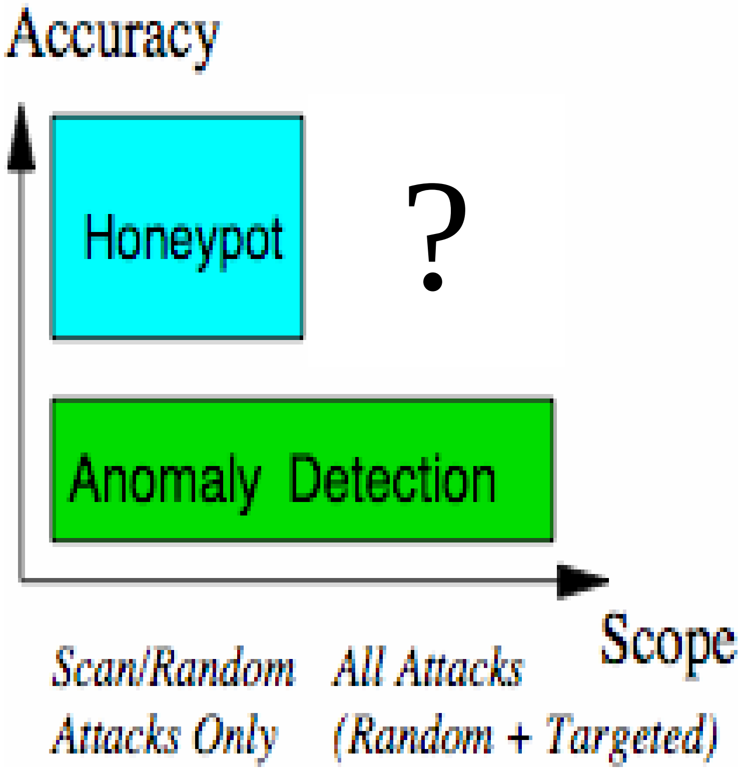
Detecting Targeted Attacks Using Shadow Honeypots

Kostas G. Anagnostakis

P. Akritidis, K. Xinidis, E. Markatos, A. Keromytis

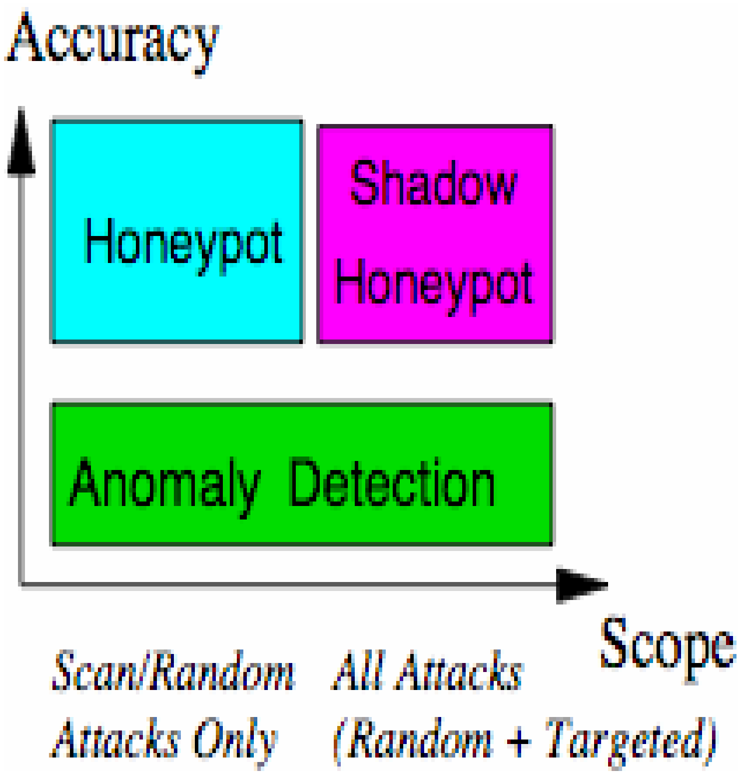
Motivation

- Malicious activity an increasing problem
- Rule-based systems prevent against known attacks
- Honeypots and ADS’s can detect zero-day attacks
  - Tradeoff between accuracy and scope

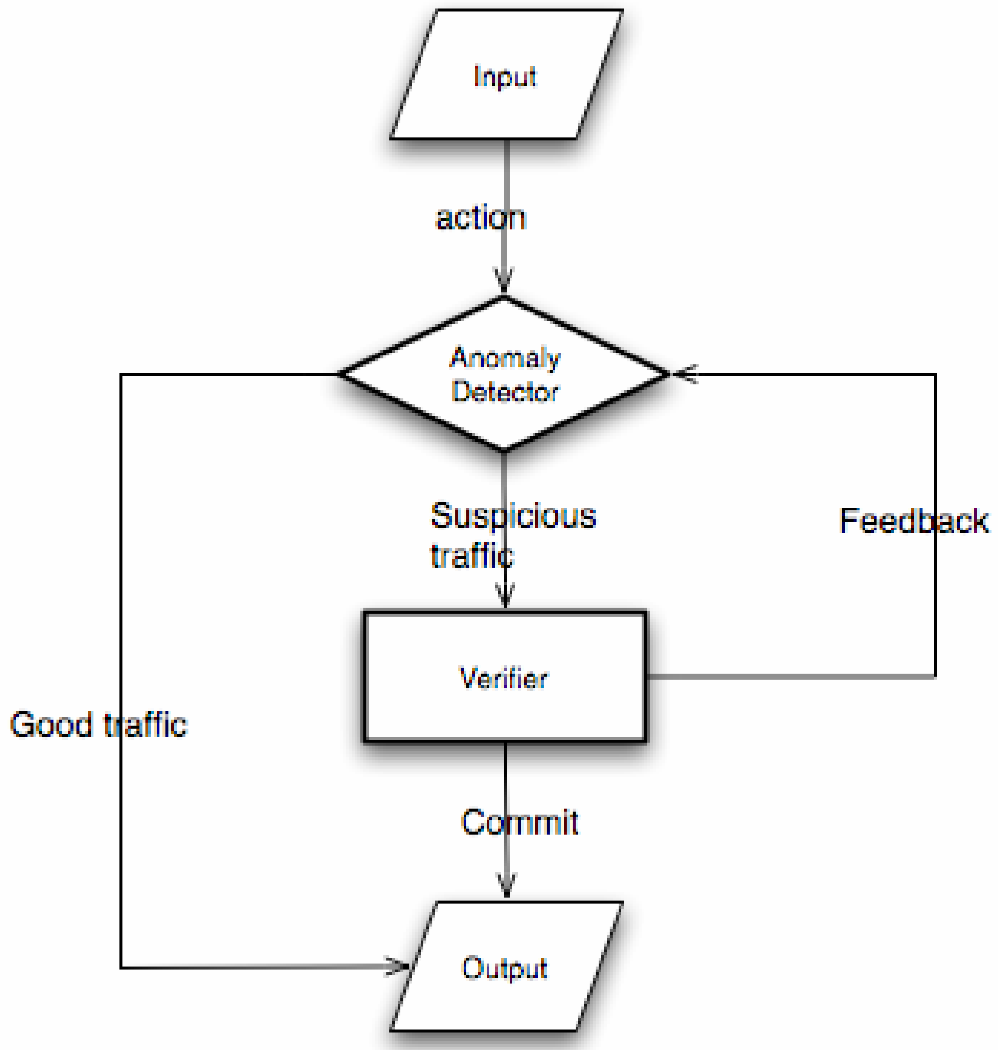


Contributions

- Hybrid model best of both worlds
  - Tune ADS for low False Negatives (FN)
  - False Positives (FP) weeded out by shadow honeypot
- Defend against targeted attacks
- Protect against client-side attacks

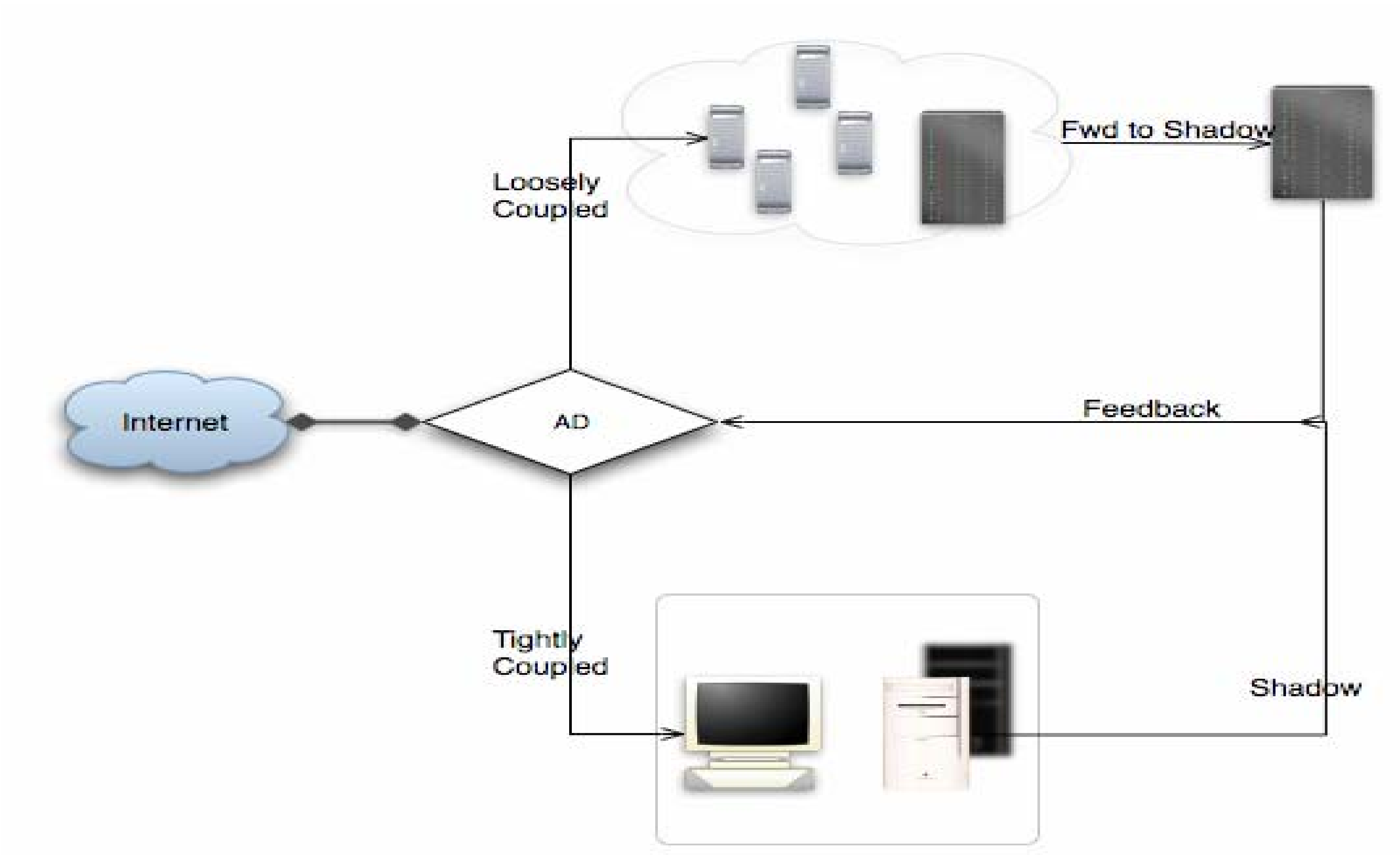


Overview

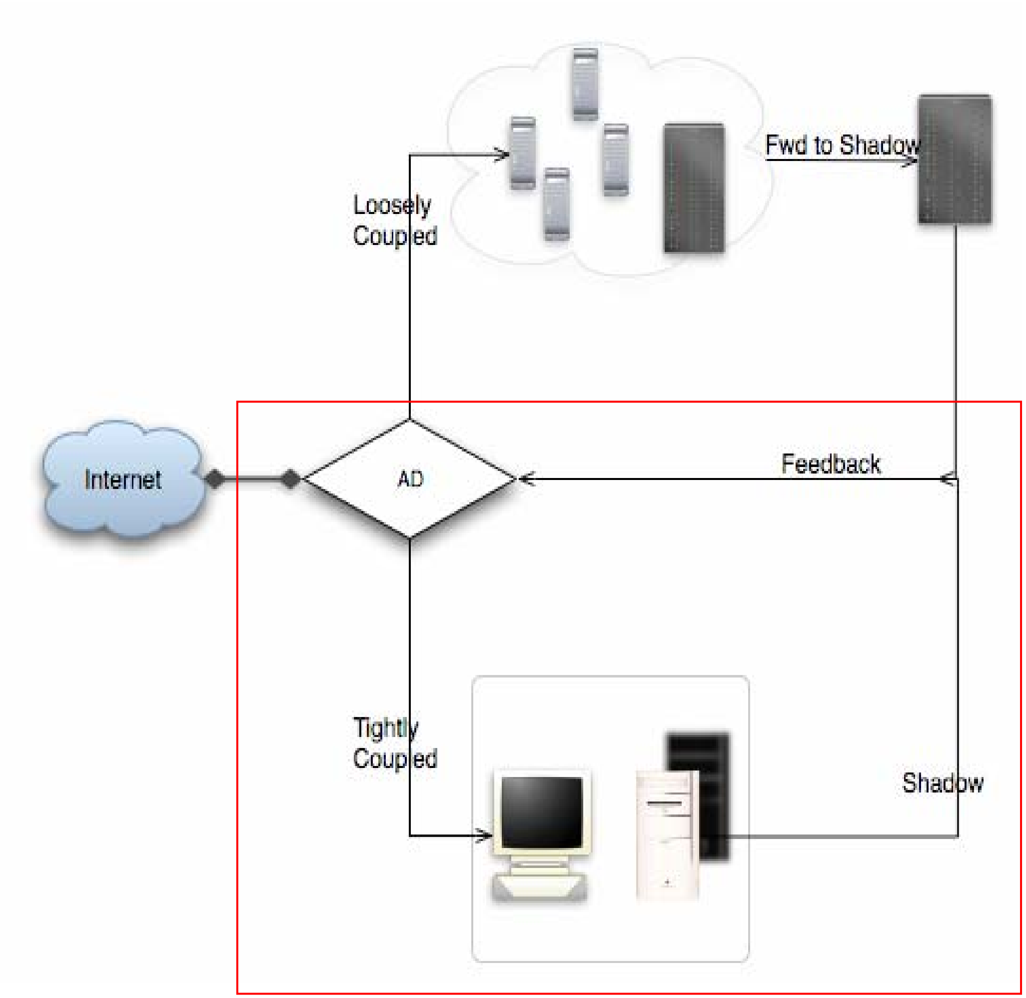


Input
action
Anomaly Detector
Suspicious traffic
Feedback
Verifier
Good traffic
Commit
Output

Tightly & loosely coupled



Tightly coupled



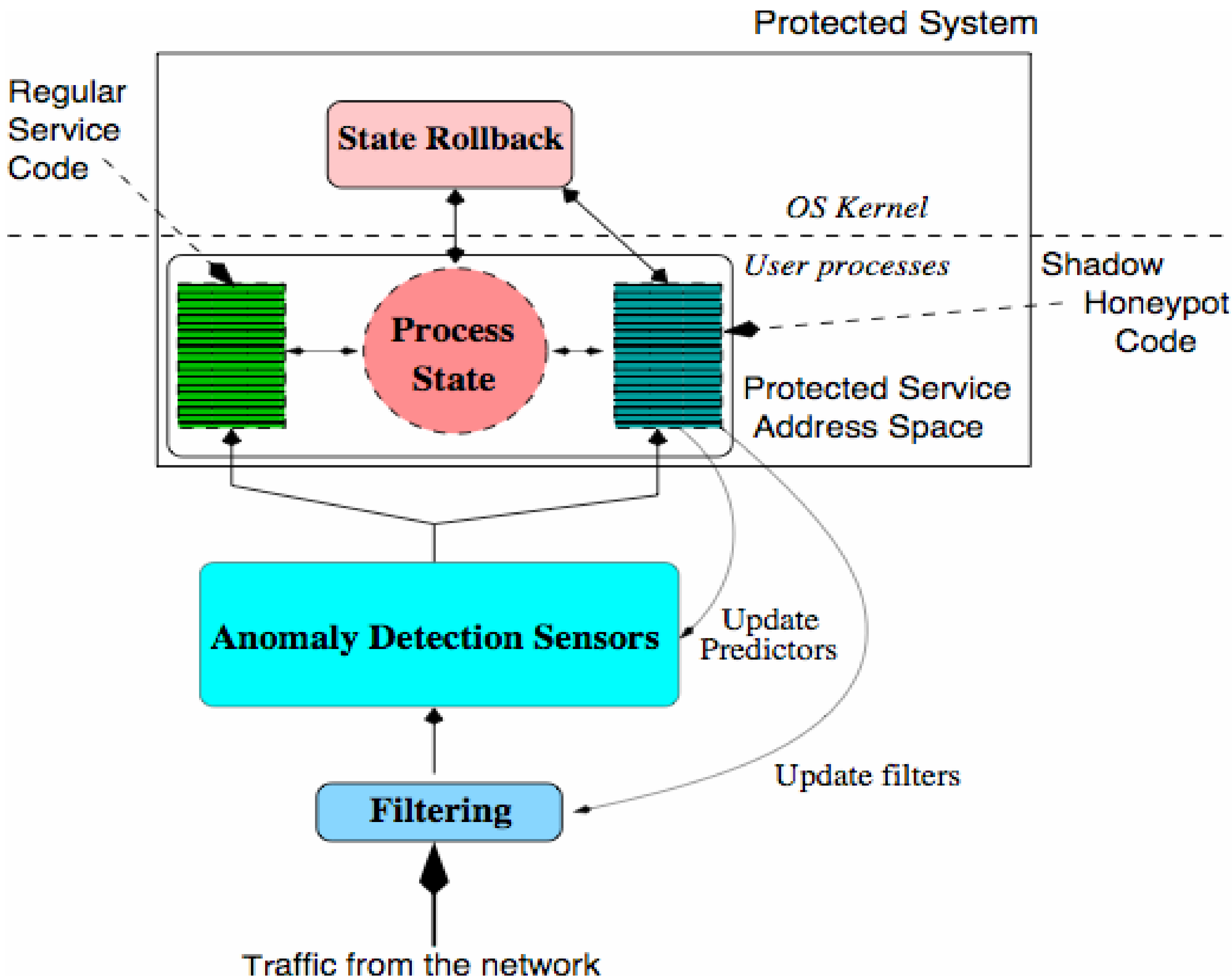
With Server

- Most practical scenario
- Fwd suspicious requests to its shadow
- Mirror functionality and state (same address space)

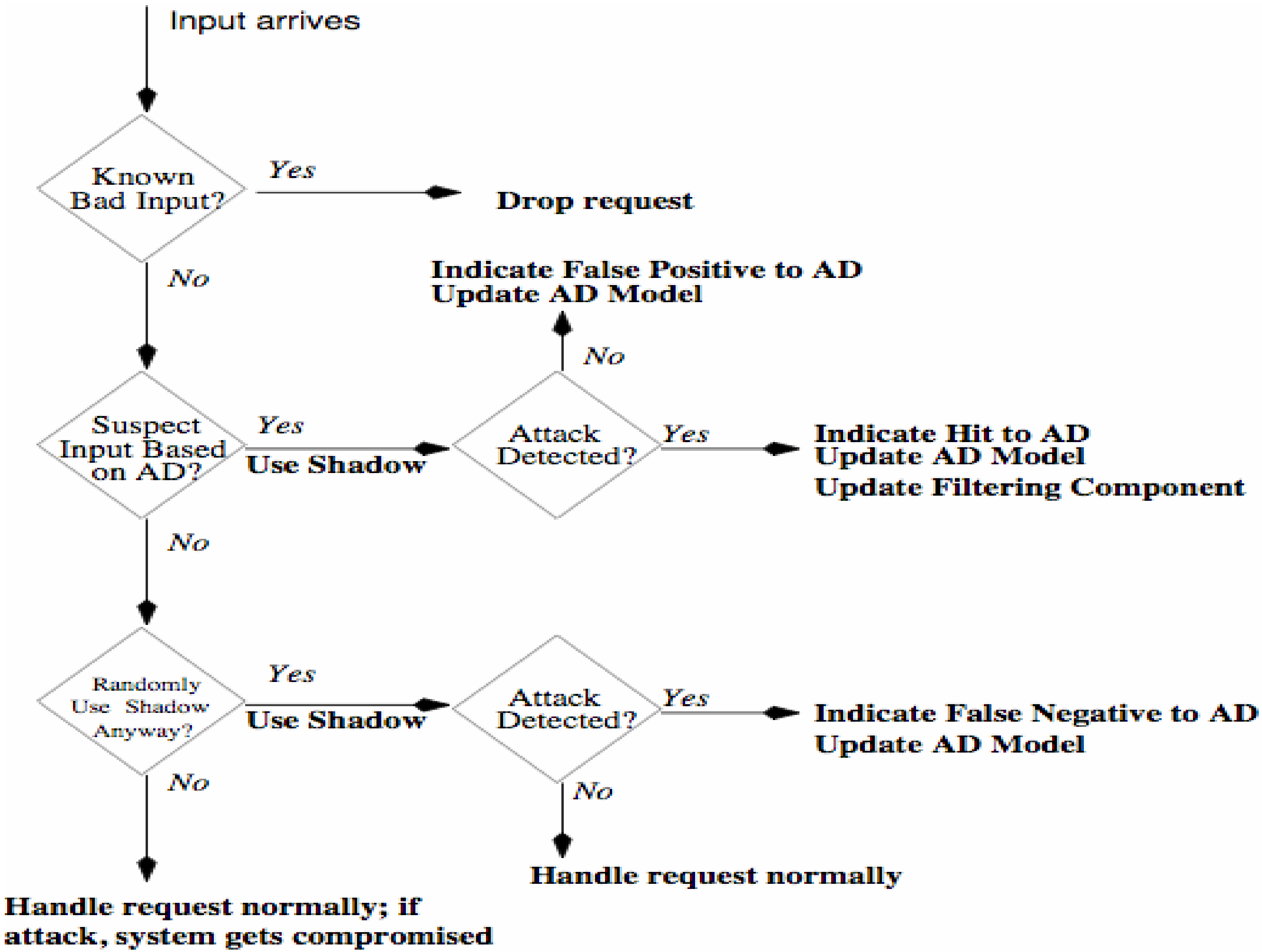
With Client

- Targets passive attacks
  - Download data containing an attack
- Shadow version of the application is invoked

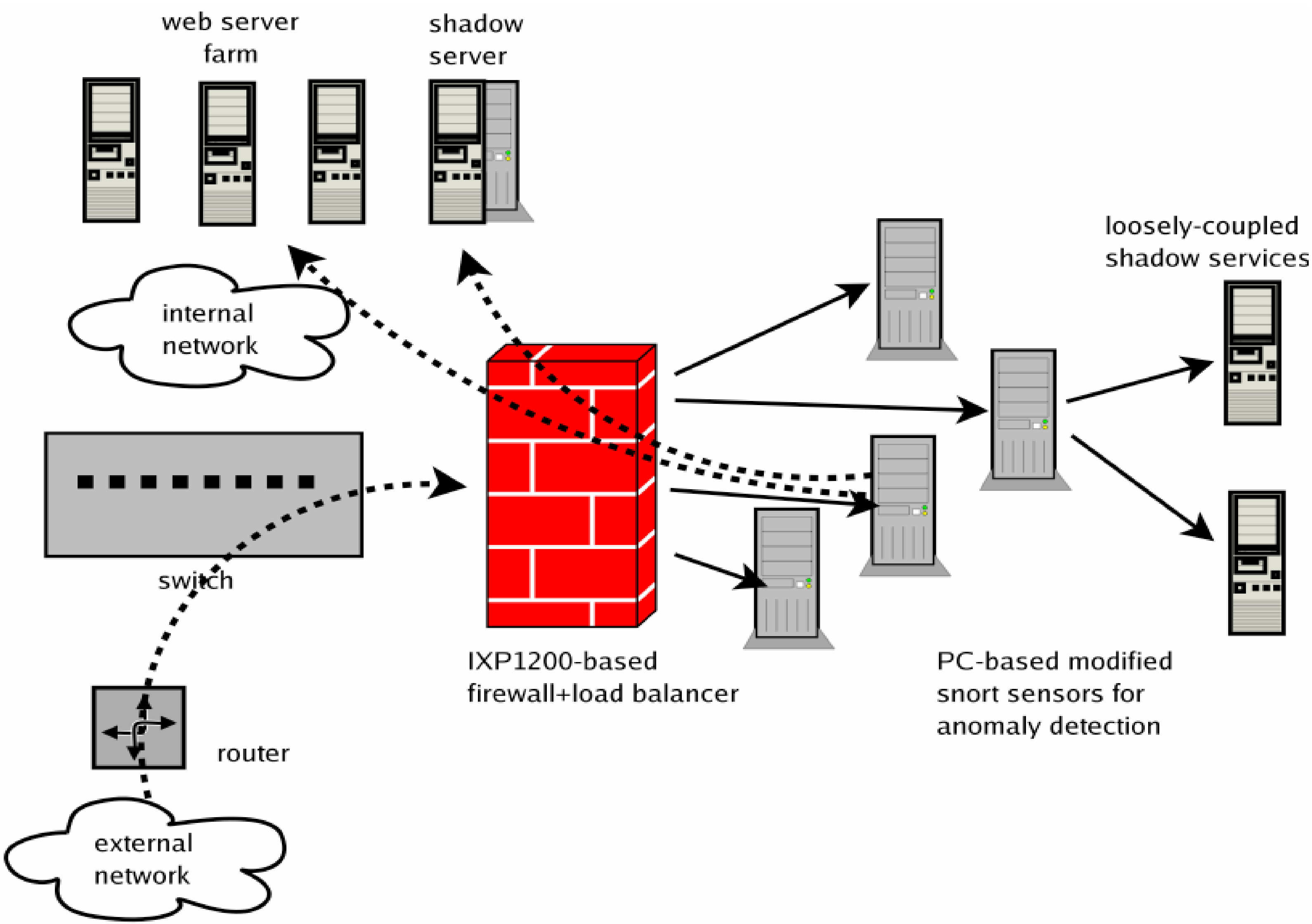
Tightly coupled architecture



System Workflow

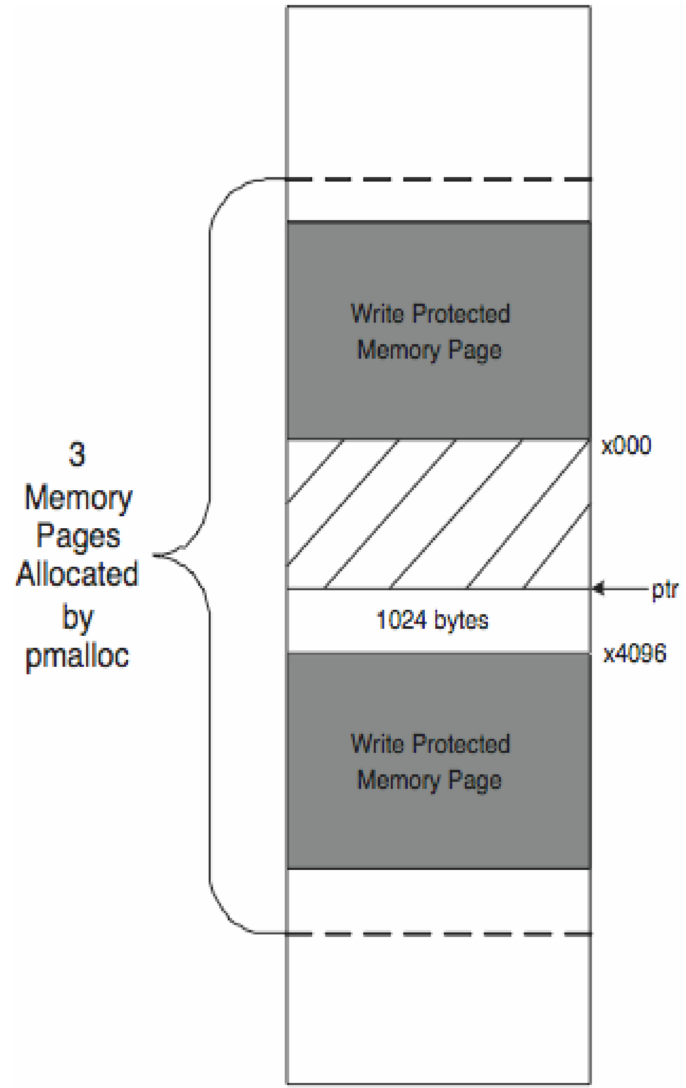


Prototype Implementation



Shadow Honeypot Creation

- Use DYBOC code transformation tool
  - Could use STEM
  - “Building a Reactive Immune System for Software Services” USENIX 05
- Focus on memory violations (buffer overflows)
- Selectively turn on defense



Shadow generation

- Original code

```
int func(char *tmp) {
  char buf[10];
  ...
  strcpy(buf, tmp);
  ...
}
```

- Transformed Code

```
int func(char *tmp) {
  char *buf;
  char buf_tx1[10];
  if (shadow_enable())
    buf = pmalloc(10);
  else
    buf = buf_tx1;
  ...
    strcpy(buf, tmp);
  ...
   if (shadow_enable())
      pfree(buf);

}
```

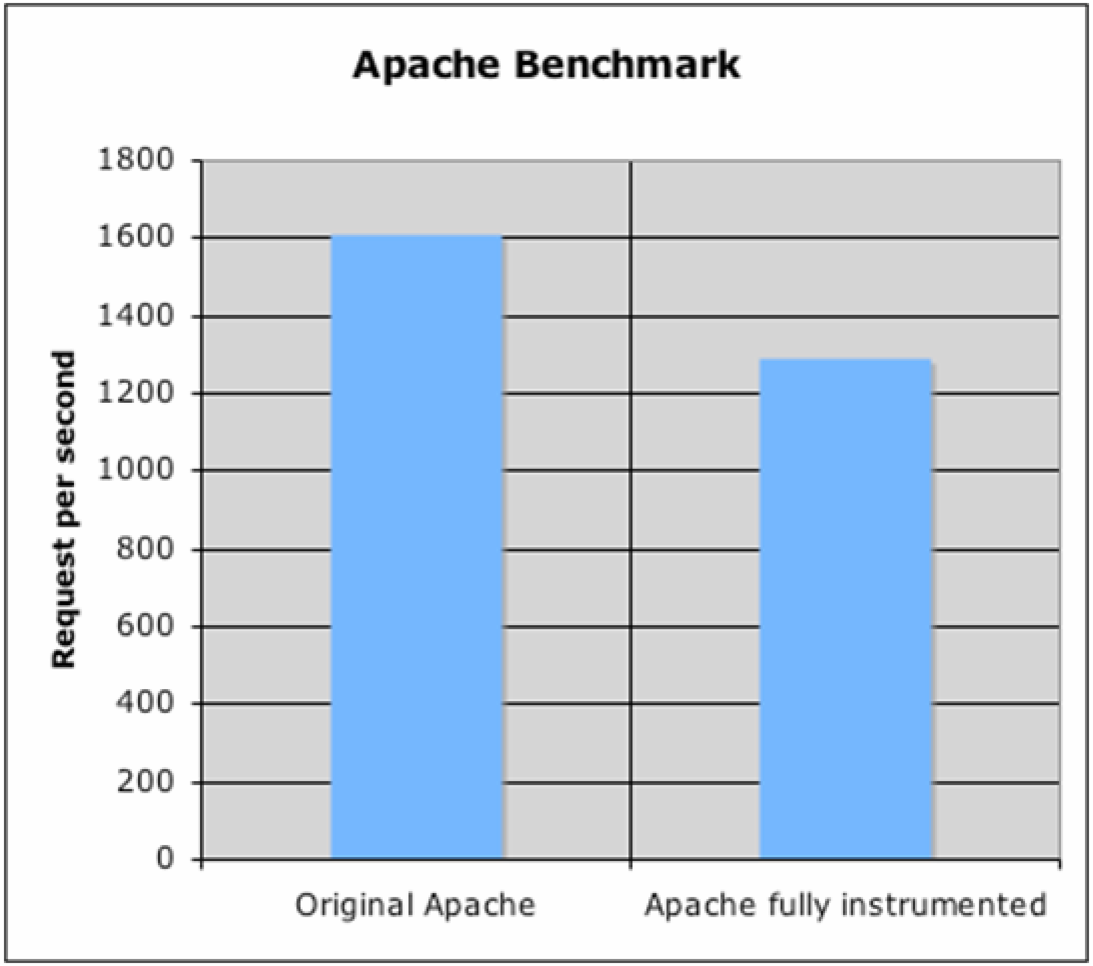
State Rollback

- New system call : *transaction()*
- Selectively invoked through shadow_enable()
  - Main processing loop, before new request
    - Indicate new transaction has begun
    - Backup memory pages (read only)
    - New page for modified pages
    - Similar to copy-on-write
  - Main processing loop, end of request
    - Indicate transaction successfully completed
    - Discard original memory pages that were modified
  - Signal handler
    - Indicate that an attack has been detected
    - Discard modified memory pages, restore original
- Filesystem

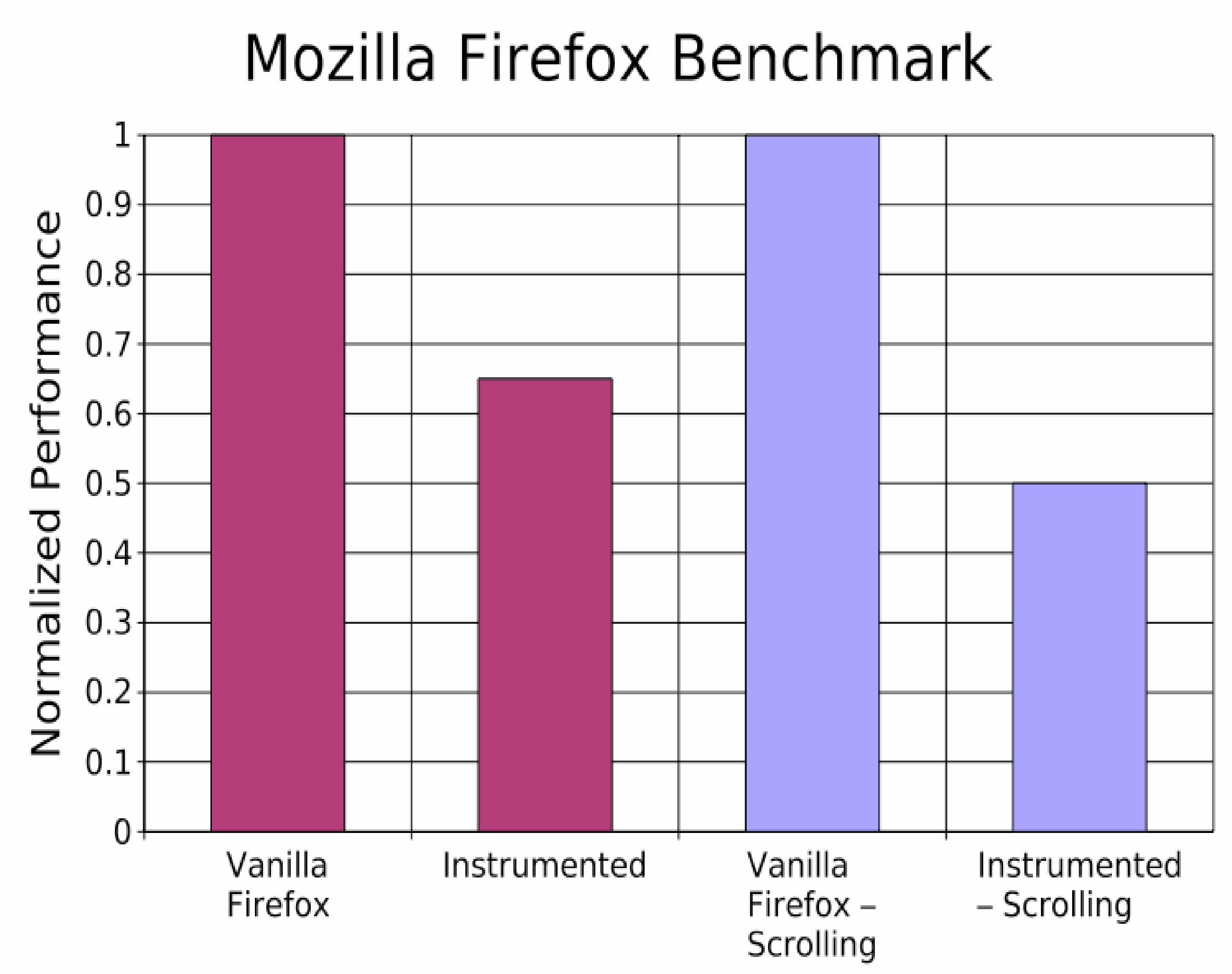
Overall system performance

- Workload generated by AD to shadow
  - False positives
  - Used real traces
    - 20-30 FP/sec
- Cost of processing by Shadow Honeypot
  - Server-side: Apache
  - Client-side: Firefox

Shadow Performance - Server



Shadow Performance - Client



Limitations and open issues

- Overload attacks on the system
  - High volume attacks
  - High volume FPs
- Multi-step attacks & root-cause isolation
- Effectiveness of training depends on quality of the anomaly detector

Potential Future Work

- Evaluate different components
  - Use STEM, taintcheck, MINOS
  - Different anomaly detection algorithms
  - Tune AD heuristics using feedback
- Experiment with loosely-coupled shadow honeypots
  - Integration with passive mon infrastructure
  - Focus on “high-risk” applications (file sharing, …)

Summary

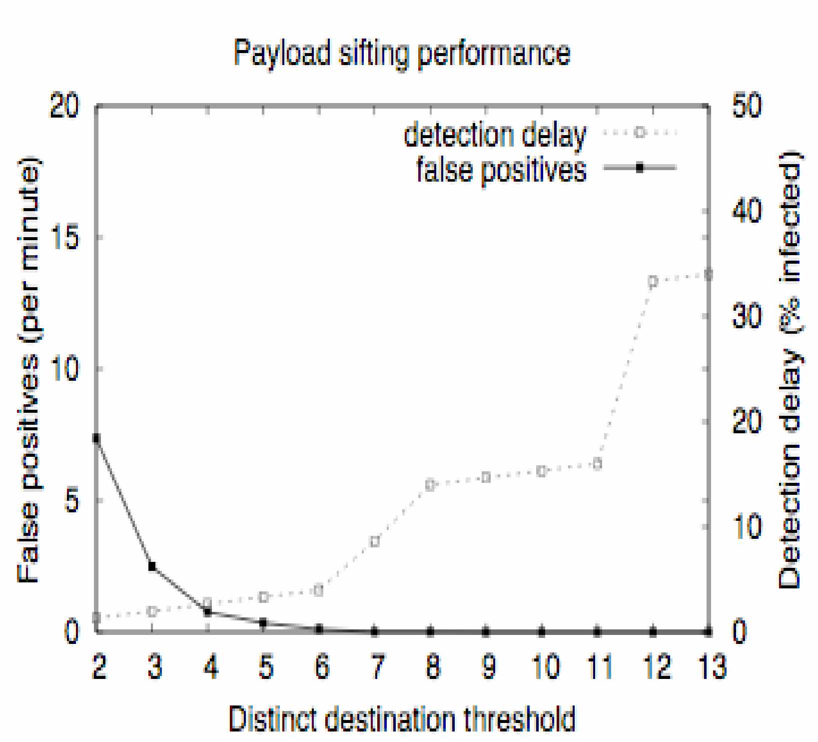
- New hybrid approach to dealing with targeted attacks by combining features found today in honeypots and ADS
- Proof of concept implementation
  - Capable of sustaining workload of protected services

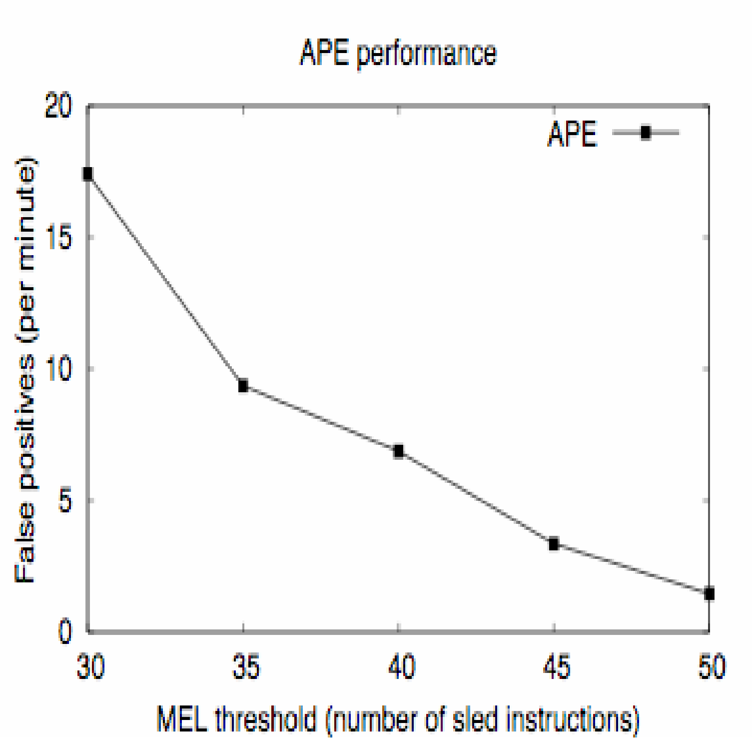
PC-based sensor performance

Detection method	Throughput/sensor
Content matching	225 Mbit/s
APE	190 Mbit/s
Payload sifting	268 Mbit/s

Filtering & AD performance

- False positive vs detection rate





Filtering - IXP1200 Utilization