Protecting against DoS Attacks by Analysing the Application Stress Space

Cornel Barna[1]

Chris Bachalo[2]

Marin Litoiu[1]

Hamoun Ghanbari[1]

Mark Shtern[1]

[1]Computer Science and Engineering
York University

[2]Juniper Networks

February 1, 2015

Motivation

- in the last years, there is a rise of DoS attacks on internet (RIAA, MPAA, Operation Payback, attack on North Korea, etc.);
- attackers developed easy-to-use toolkits: Low Orbit Ion Cannon (LOIC);
- motivation behind the attacks:
  - financial;
  - political;
  - ideological;
  - it's "cool" to do it.

DoS Attacks

Network Protocols were designed to be functional, not secure.

- flood the system with requests;
- designed to make the system unresponsive;
- legitimate users cannot receive service;
- methods to fix the problem:
  - add more and better hardware;
  - use a firewall to filter malicous traffic.
- it's a challenge to defend against them;
- the current state of the art methods do not fully mitigate the DoS attacks.

Types of Attacks

- Regular Denial of Service (DoS)
  - attacker uses one computer to launch the attack
- Distributed Denial of Service (DDoS)
  - the attacker uses multiple computers
- Reflected Denial of Service (RDoS)
  - the attacker spoofs the source address
  - uses intermediaries (reflectors) to amplify the attack
- Distributed Reflected Denial of Service (DRDoS)
  - same as RDoS, but the attacker uses multiple computers

Application Layer DoS

- targets a specific service (HTTP, HTTPS, SMTP)
- less traffic required to saturate the service
- malicious HTTP headers are not ditinguishible from legitimous ones
- HTTP POST DDOS attack
  - send the "Content-Length" header with a very large value
  - send the body of the request very slow (1 byte / 110 seconds)
- slowloris
  - keep open as many connections possible for as long possible
  - send partial requests
  - saturates the connection pool

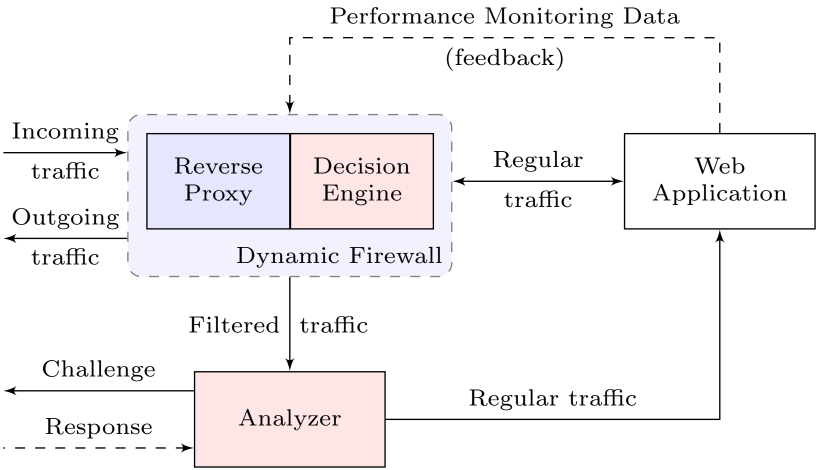
Adaptive DoS Mitigation

If the traffic exceeds our ability to manage it, there is a DoS attack.

- we created a framework to mitigate (D)DoS attacks;
  - dynamic firewall;
  - decision engine;
  - analyzer.
- statistical anomaly detection to create the filtering rules;
- a performance model fine-tunes the filtering rules;
- the protected system is continously monitored;
- traffic filtering rules are dynamicaly added/removed;

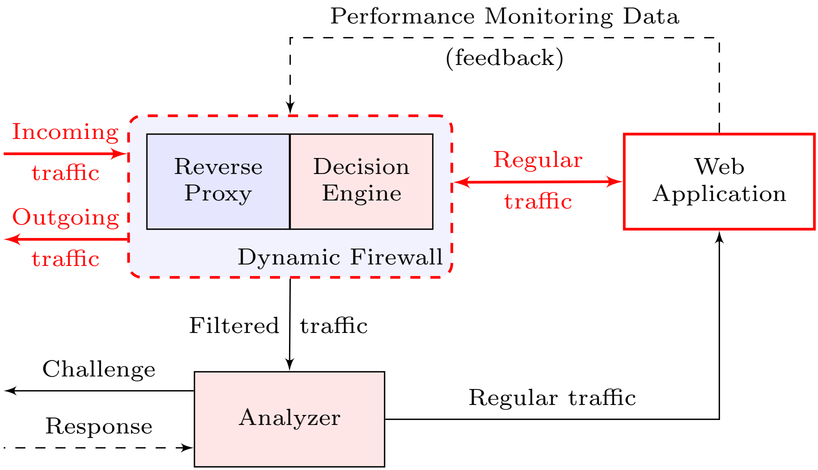
The architecture

The framework.



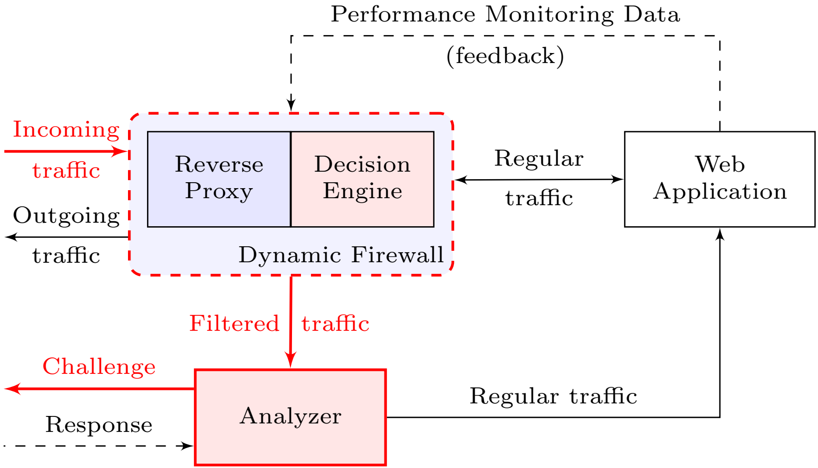
The architecture

Not a filtered request: forward it to the web application



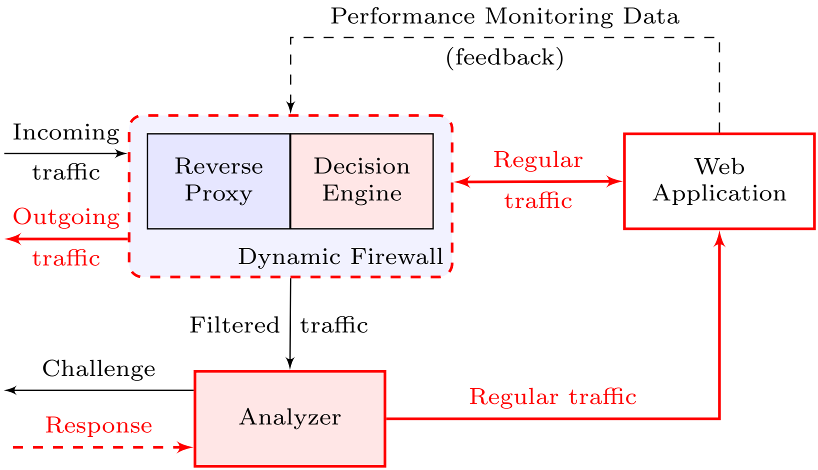
The architecture

Filtered request: issue a challenge (CAPTCHA)



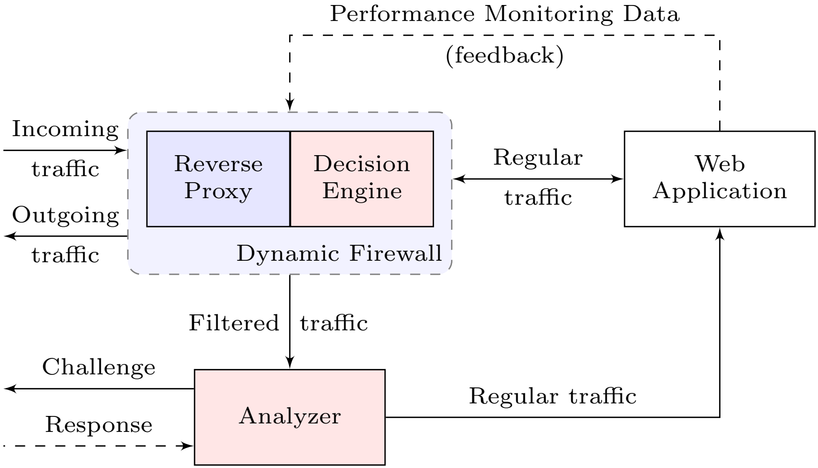
The architecture

Challenge solved: forward the request to the web application



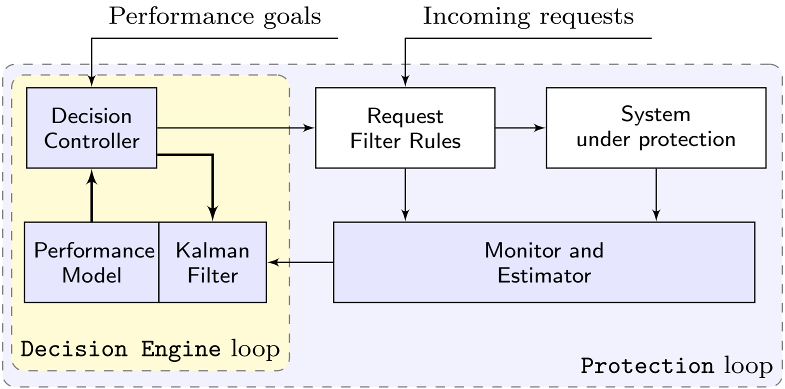
The architecture

The framework.



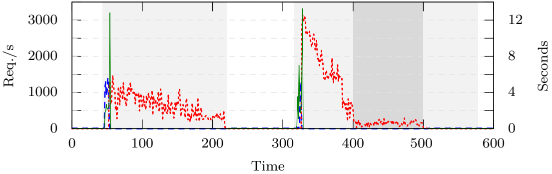
The decision engine

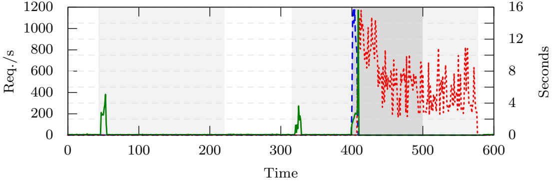
- `protection loop`: monitors the system under protection;
- `decision loop`: create/remove fitering rules.

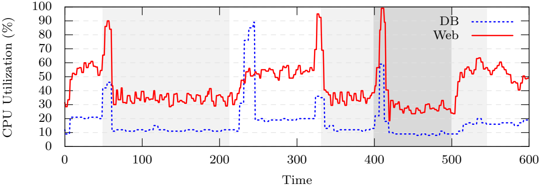


Results

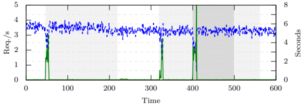
Reactive Systems

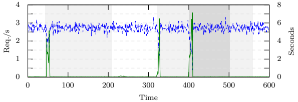


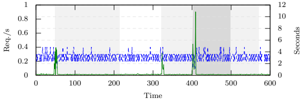


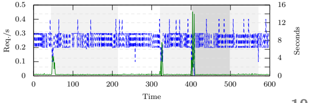


Emulated DoS attack on two scenarios using LOIC.





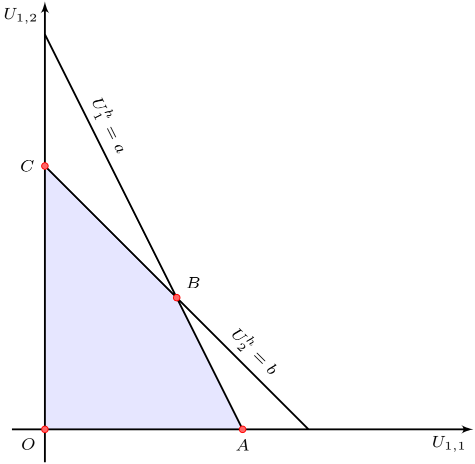




The Performance Stress Space

For proactive systems, we need better prediction!

Consider a system with two *hardware* resources, two *software* resources and two *classes of service*.



Hardware Constraints

- Linear equations:

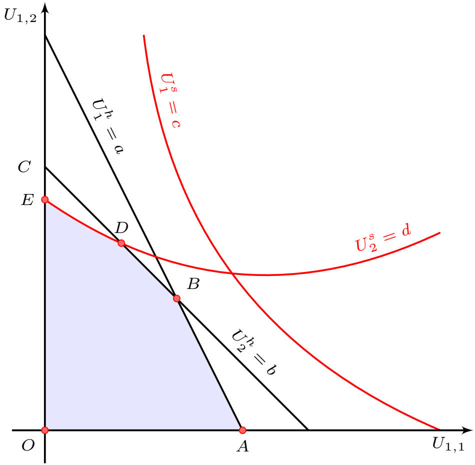
$$U_K^h = \sum_{\forall C \in \mathcal{C}} \frac{D_{K,C}}{D_{K_r,C}} \times U_{K_r,C}, \quad \forall K \in \mathcal{K}^h$$

- Feasible space $\longrightarrow$ *OABC*

The Performance Stress Space

For proactive systems, we need better prediction!

Consider a system with two *hardware* resources, two *software* resources and two *classes of service*.



Hardware Constraints

- Linear equations:

$$U_K^h = \sum_{\forall C \in \mathcal{C}} \frac{D_{K,C}}{D_{K_r,C}} \times U_{K_r,C}, \quad \forall K \in \mathcal{K}^h$$

- Feasible space $\longrightarrow$ *OABC*

Hardware and Software Constraints

- Non-linear equations:

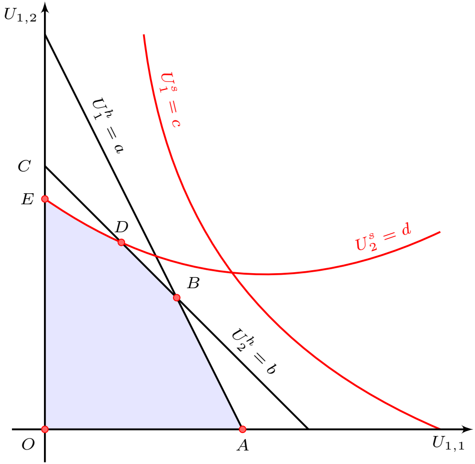
$$U_K^s = \sum_{\forall C \in \mathcal{C}} \frac{R_{K,C}^s}{D_{K_r,C}} \times U_{K_r,C}, \quad \forall K \in \mathcal{K}^s$$

- Feasible space $\longrightarrow$ *OABDE*

The Performance Stress Space

For proactive systems, we need better prediction!

Consider a system with two *hardware* resources, two *software* resources and two *classes of service*.



- system becomes saturated when close to EDBA lines $\longrightarrow$ likely to crash/deny service
- we should predict where the workload is heading and how close are we to EDBA lines
- how to calculate the shortest distance to the boundary? Perpendicular to the border lines?

Experiment

Analytically find the scenarios that should be filtered.

Insert	Update	Sel 0	Sel 1	Sel 2	Sel 3	Distanc
0.156432	0.155117	0.660674	0.013657	0.010709	0.003409	0
0	0	0.769695	0.230305	0	0	0.327842
0	0	0.949459	0	0.050541	0	0.365669
0	0	0.972853	0.006371	0	0.020775	0.382697
0	0	0.977713	0	0.001594	0.020692	0.3868
0	0	0.979214	0	0	0.020786	0.388076
0	0	1	0	0	0	0.404953
0	0	0.536736	0	0.015103	0.448161	0.511765
0	0	0.534068	0.015975	0	0.449957	0.513894
0	0	0.521225	0	0.478775	0	0.535968
0	0	0.42254	0.57746	0	0	0.650569
0.224242	0	0	0	0.465403	0.310355	0.875385
0.194327	0	0	0.428746	0	0.376927	0.879726
0	0.17826	0	0	0.429394	0.392346	0.887836
0	0.15609	0	0.399698	0	0.444211	0.89689
0	0.65056	0	0.34944	0	0	0.905153

- Web Application with 6 scenarios
- first row—user ratio when the attack started
- 2+ rows—stress vectors (46 total, only showing a few)
- last column—euclidian distance between the stress vectors and user ratio at the moment of attack

Conclusion

- use a predictor for workload and identify system overload before it happens
- challenges:
  - prediction is hard
  - how close is a given workload to the edges of feasible space?
  - how do we validate the method?
- benefits for very large scale systems: reduce the search space to find
  - when the system will saturate/crash
  - what scenarios to filter
  - what types resources to add
  - dynamic decision making (not development time)

1. How close is a given workload to the edges of feasible space? This need to be found automatically, at runtime.
2. What is the interpretation for the distance to the edges?
3. Could the feasible space be useful to solve other problems?