DDoS Defense by Offense

Michael Walfish, Mythili Vutukuru,
Hari Balakrishnan, David Karger,
Scott Shenker , SIGCOMM '06

Presented by Lianmu Chen

Outline

- Introduction
- Design
- Implementation
- Evaluation
- Conclusions

Introduction

Introduction

- Application level DDoS – It is a noxious attack in which computer criminals mimic legitimate client behavior by sending proper-looking requests, often via compromised and commandeered hosts known as *bots.*

- Attacker sends proper looking requests to waste server's resources; Overwhelms server, not access links.

Introduction

- Far less bandwidth is required: the victim's computational resources—disks, CPUs, memory, application server licenses, etc.—can often be depleted by proper-looking requests long before its access link is saturated.

- The attack traffic is "in-band," it is harder to identify and thus more potent.

Three categories of Defenses

- Overprovision computation resources massively

- Detect and block

- *Resource-based* defenses

Speak-up

- It's a Resource-based defense that uses *bandwidth* as the currency.
  - Claim: attackers use most of their available bandwidth during attacks, victims do not.
  - Use *encouragement* to make victims send more traffic so they are better represented at the server.

Threat Model ???

- The attacker can send difficult requests intentionally.
- An attacker can repeatedly request service from a site while having different IP addresses.

Two conditions to make it work

- **Adequate Client Bandwidth**: the good clients must have in total roughly the same order of magnitude (or more) bandwidth than the attacking clients.
- **Adequate Link Bandwidth**: The protected service needs enough link bandwidth to handle the incoming request stream.

Three conditions where it wins

- **No predefined clientele:** otherwise the server can install filters to permit traffic only from known clients.
- **Non-human clientele**: ruling out proof-of humanity tests.
- **Unequal requests *or* spoofing *or* smart bots**: Currency based approach can charge clients for harder requests.

Design

Speak-up

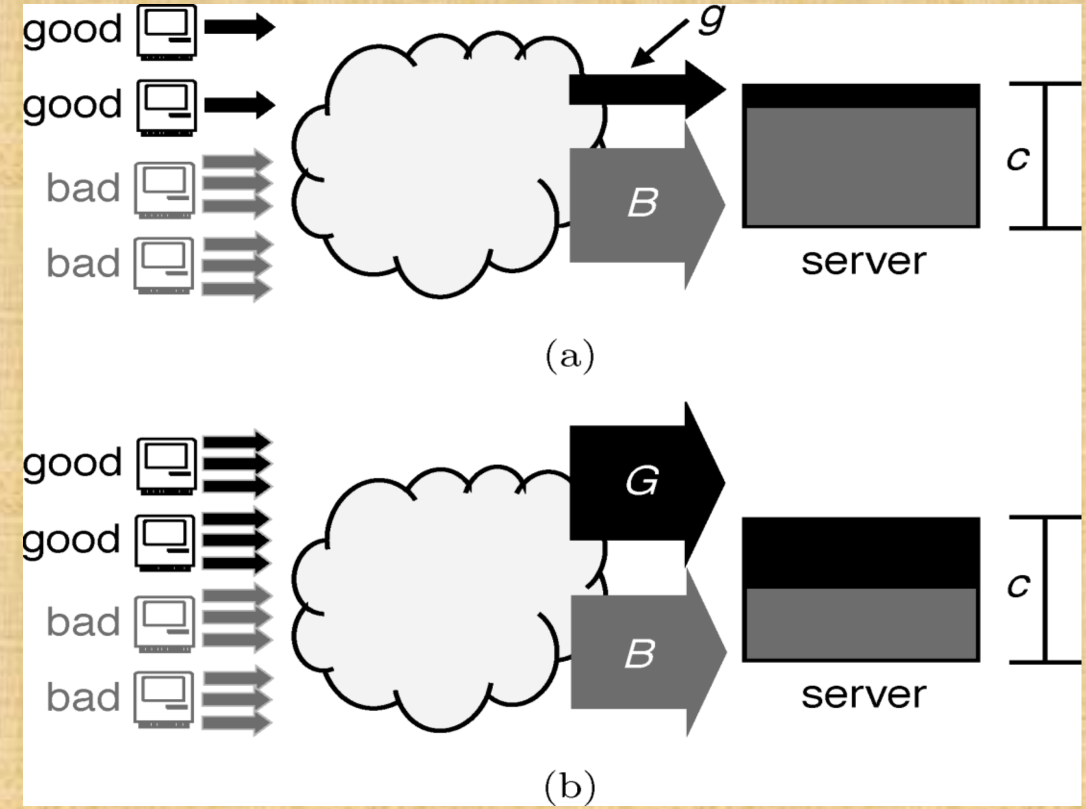


Illustration of speak-up

(a) g/g+B (b) G/G+B

-Bad clients exhaust all of their available bandwidth on spurious requests.

-Good clients are likely using a only small portion of their available bandwidth.

-The key idea of speak-up is to exploit this difference.

Design Goal

Allocate resources to competing clients in proportion to their bandwidth.

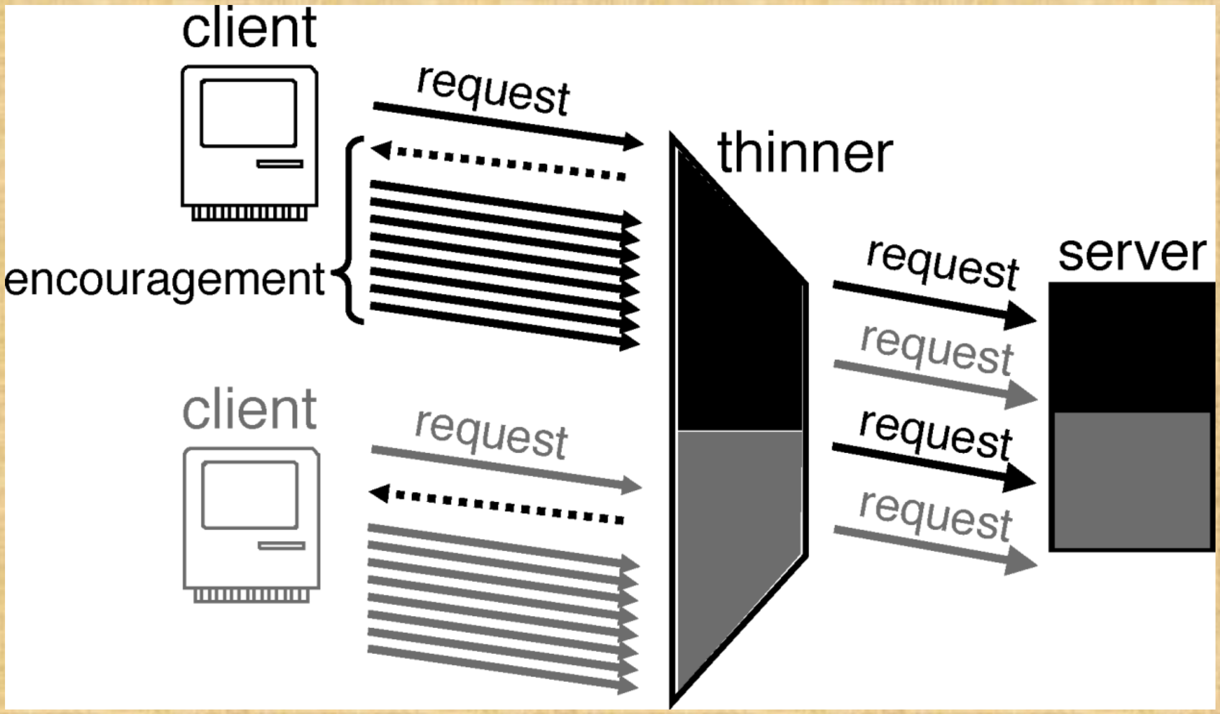
- If the good clients make g requests per second and have an aggregate bandwidth of G requests per second to the server and if the bad clients have aggregate bandwidth of B requests per second then the server should process good requests at a rate of min(g,(G/G+B)c) requests per second where c is the servers capacity to process requests.

Required mechanisms

- Limit the requests to a server to c per second.
- Perform encouragement : cause a client to send more traffic.
- Speak-up needs a proportional allocation mechanism to admit client at rates proportional to their delivered bandwidth.

Hence, the ***thinner*** appears.

Thinner



Under speak-up, these mechanisms are implemented by a front-end to the server, called the *thinner.*

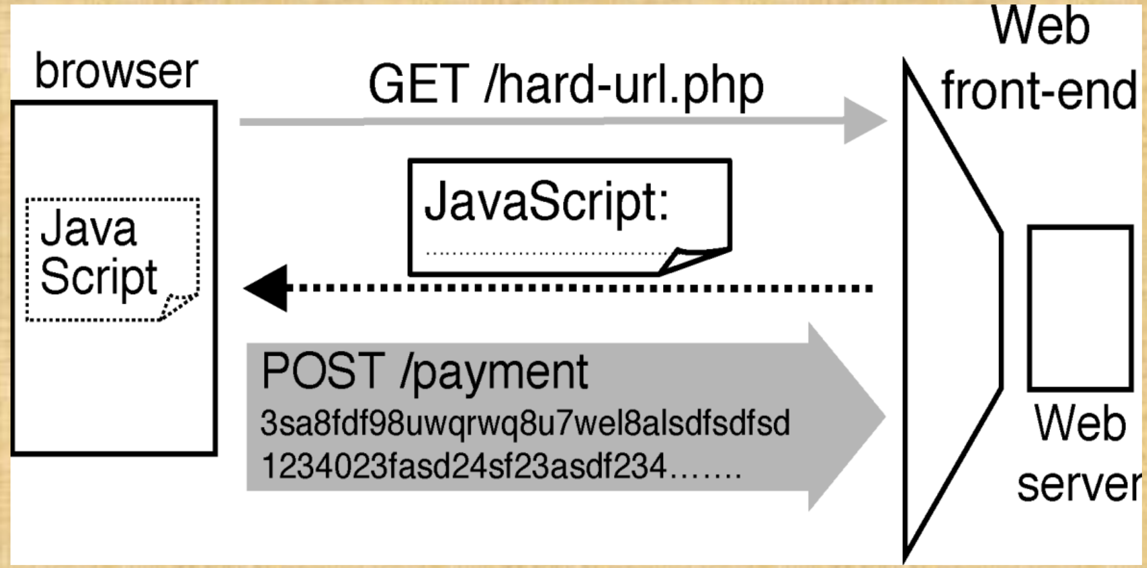
Thinner: the thinner implements encouragement and controls which requests the server sees.

Explicit Payment Channel

- When server is overloaded, thinner asks clients to open separate payment channels.
- Client sends dummy bytes on this channel, becomes a contender.
- Thinner tracks how much each contender sends.
- When the server notifies the thinner it is ready to fire a new request, thinner admits the client which has sent the most number of padded dummy bytes.

Implementation

Implementation



- **A prototype thinner is implemented in C++.**
- **It runs on Linux 2.6 exporting a well know URL.**
- **When a web client requests this URL then thinner decides , if and when to send this request to the server.**
- **When the server responds to that request, the thinner returns HTML to the client with that response.**

Implementation

- Clients send by Poisson process with limited windows (open requests).
- Deterministic service time (all requests equal)
- Bad clients send faster, and have bigger windows.
- Good client: λ = 2, w = 1
- Bad client: λ = 40, w = 20
- Max. number of clients limited to 50 by testbed.

Configuration parameters

—the *capacity of the protected server, expressed in requests per second.*

—a *list of URLs and regular expressions that correspond to "hard requests."* Each URL and regular expression is associated with a *difficulty level.*

—the *name or address of the server.*

—a custom *"please wait" screen that humans will see while the server is working* and while their browser is paying bits.

Implementation

The Web client requested a “hard” URL(HTTP GET request), the thinner replies with the “please wait”.

- no other connections to the thinner, thinner returns to the client (1) JavaScript that wipes the “please wait” screen (2) the contents of the server’s reply.
- other clients are communicating with the client submit, a one-megabyte HTTP POST containing random bytes.

  --The client wins an auction, the thinner terminates the POST and submits the client’s request to the server.

  --The client does not win, then the thinner returns JavaScript that causes the browser to send another POST, and the process described in the previous paragraph repeats.

Evaluation

Validating the thinner's allocation

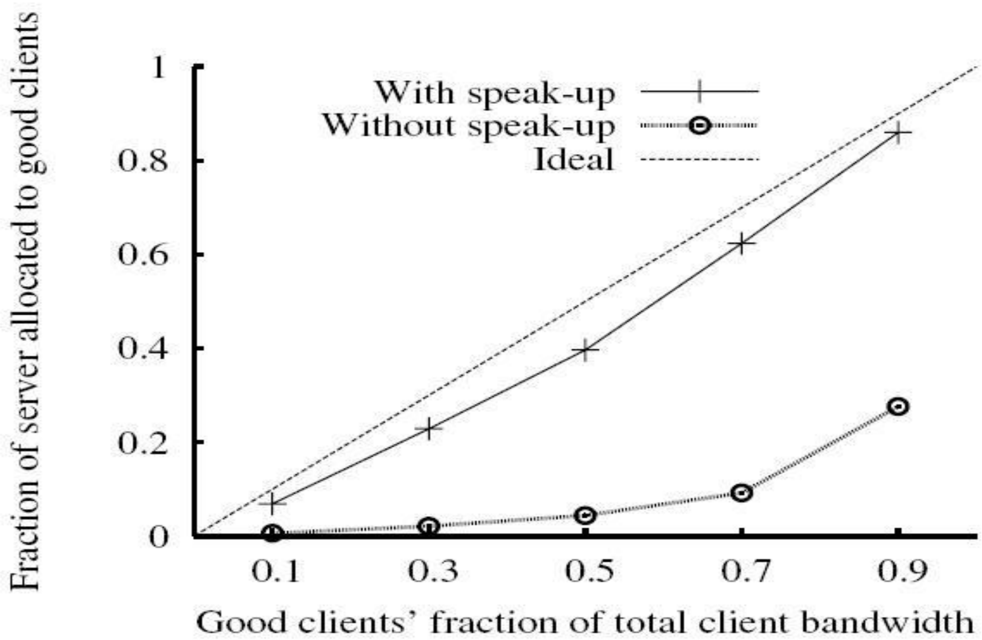


Figure 2: Server allocation when $c = 100$ requests/s as a function of $\frac{G}{G+B}$. The measured results for speak-up are close to the ideal line. Without speak-up, bad clients sending at $\lambda = 40$ requests/s and $w = 20$ capture much more of the server.

Validating the thinner's allocation

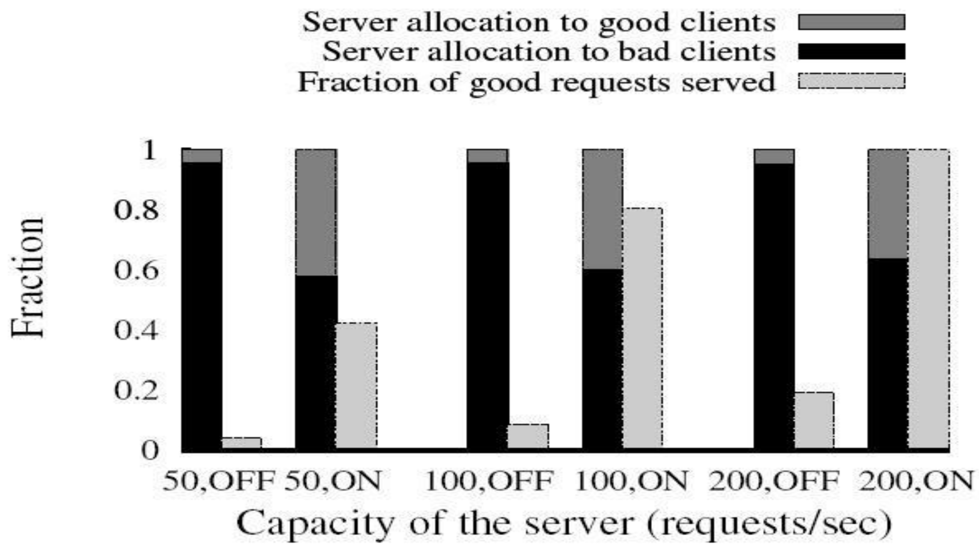


Figure 3: Server allocation to good and bad clients, and the fraction of good requests that are served, without ("OFF") and with ("ON") speak-up. c varies, and $G = B = 50$ Mbits/s. For $c = 50, 100$, the allocation is roughly proportional to the aggregate bandwidths, and for $c = 200$, all good requests are served.

Setup: 25 good clients, 25 bad clients

Cid = 100 *c* = 50, 100, 200

Latency cost

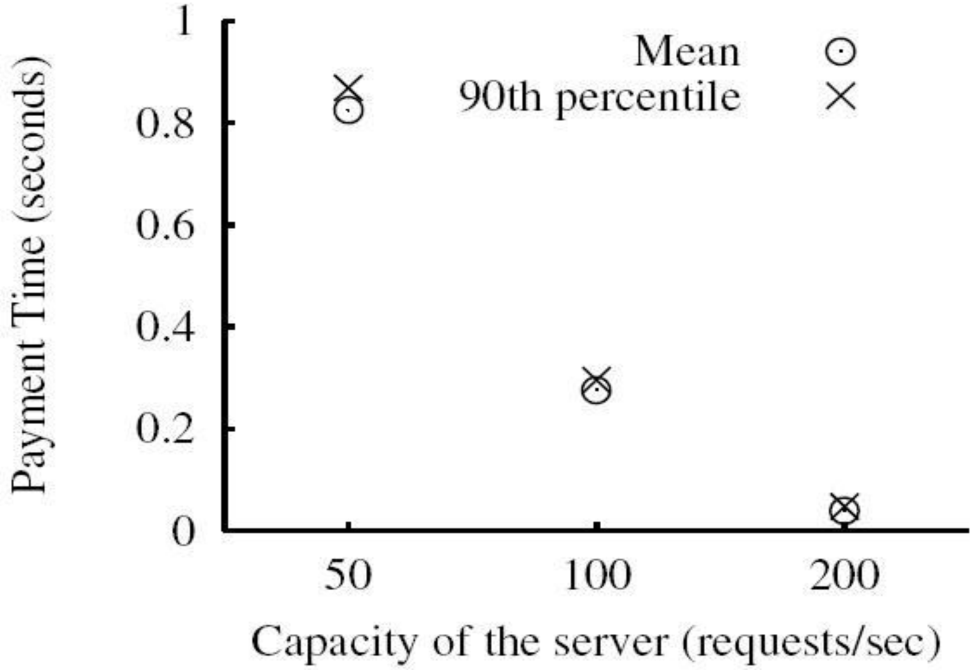


Figure 4: Mean time to upload dummy bytes for good requests that receive service. c varies, and $G = B = 50$ Mbits/s. When the server is not overloaded ($c = 200$), speak-up introduces little latency.

Byte Cost？ ？ ？

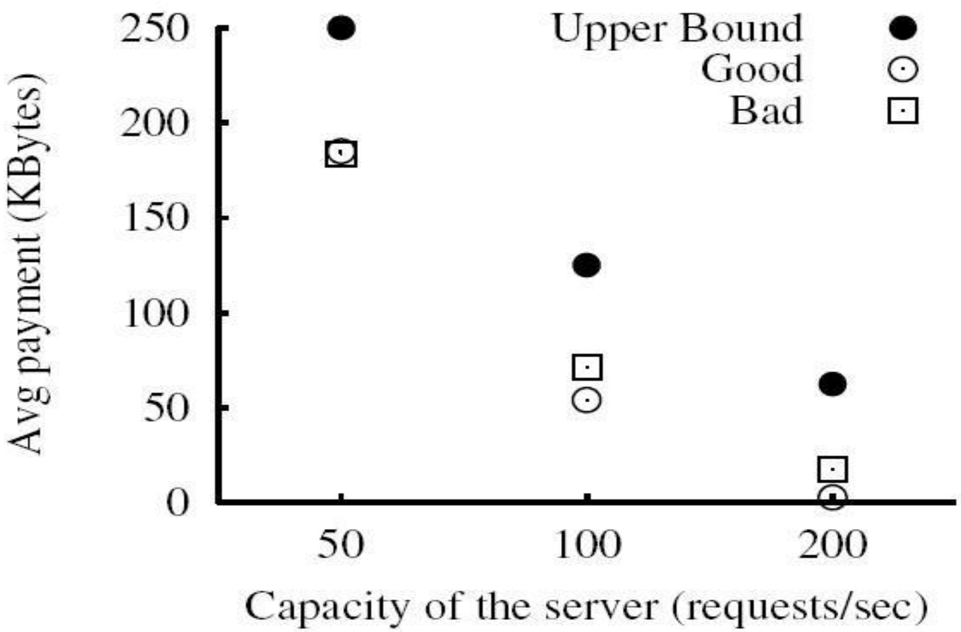


Figure 5: Average number of bytes sent on the payment channel—the “price”—for served requests. c varies, and $G = B = 50$ Mbits/s. When the server is overloaded ($c = 50, 100$), the price is close to the upper bound, $(G + B)/c$; see the text for why they are not equal. When the server is not overloaded ($c = 200$), good clients pay almost nothing.

Heterogeneous Network Conditions

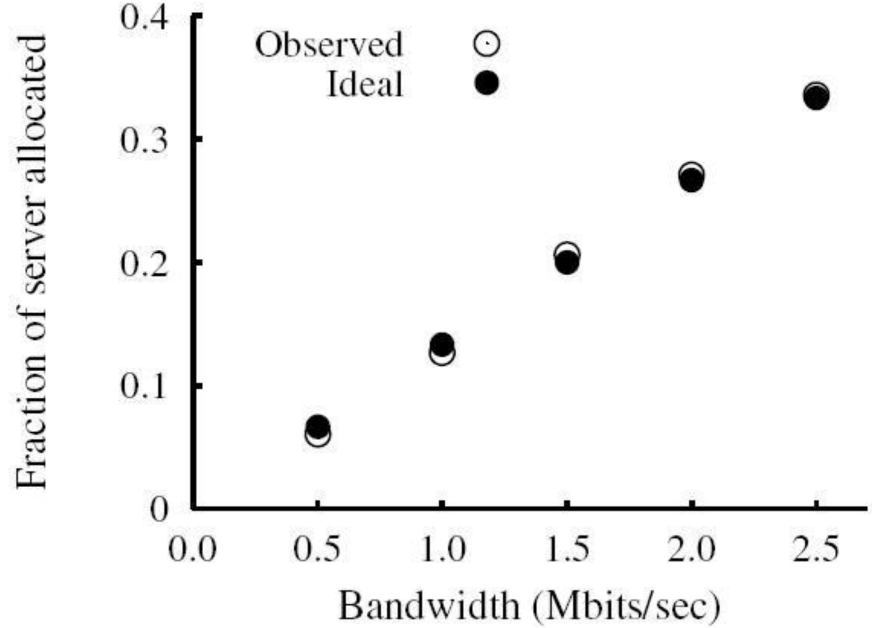


Figure 6: Heterogeneous client bandwidth experiments with 50 LAN clients, all good. The fraction of the server (c = 10 requests/s) allocated to the ten clients in category i, with bandwidth $0.5 \cdot i$ Mbits/s, is close to the ideal proportional allocation.

Heterogeneous Network Conditions

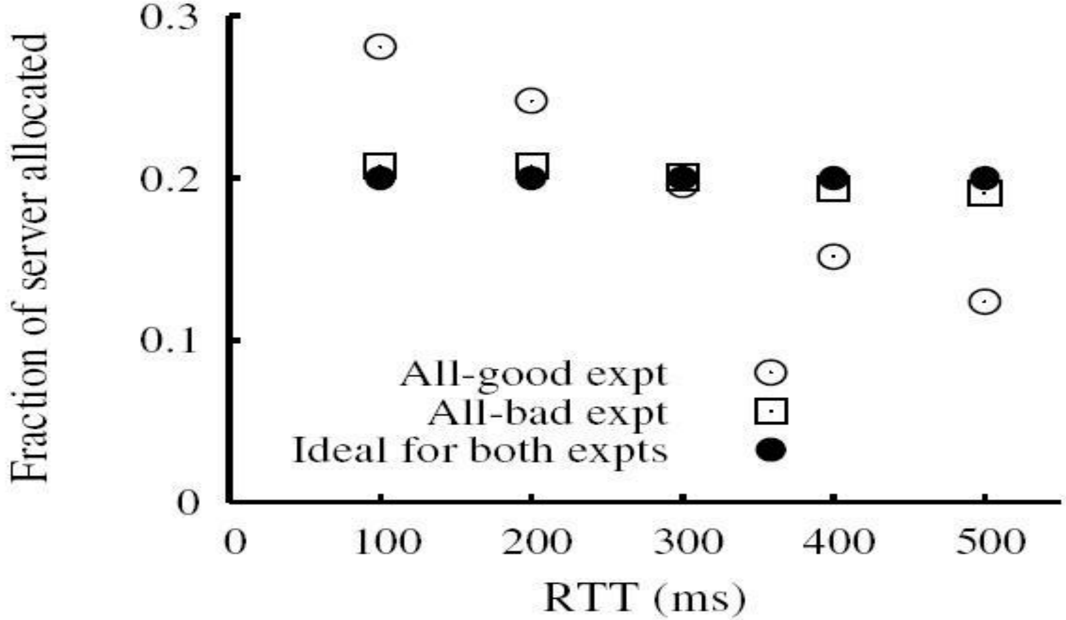
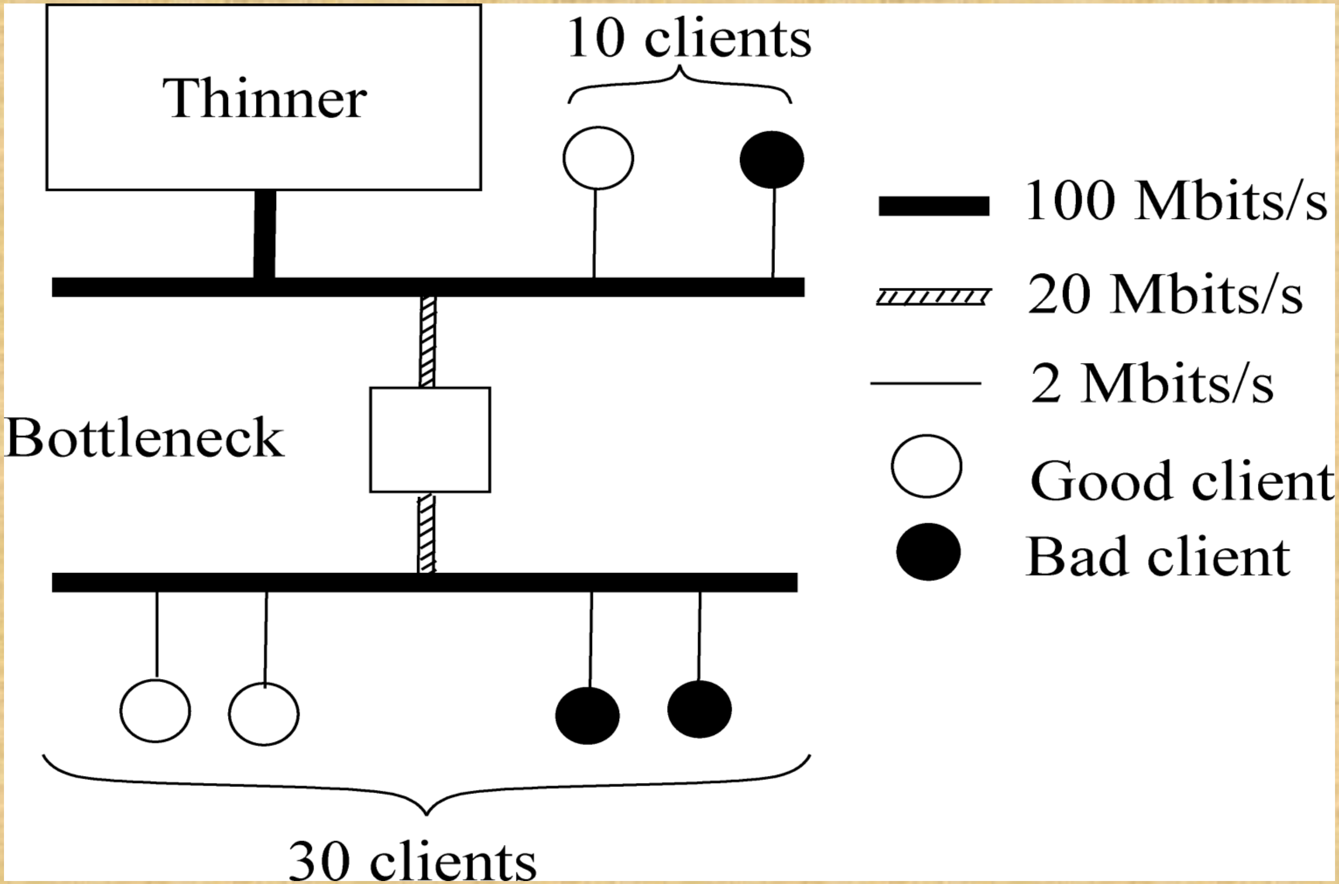


Figure 7: Two sets of heterogeneous client RTT experiments with 50 LAN clients, all good or all bad. The fraction of the server (c = 10 requests/s) captured by the 10 clients in category i, with RTT $100 \cdot i$ ms, varies for good clients. In contrast, bad clients' RTTs don't matter because they open multiple connections.

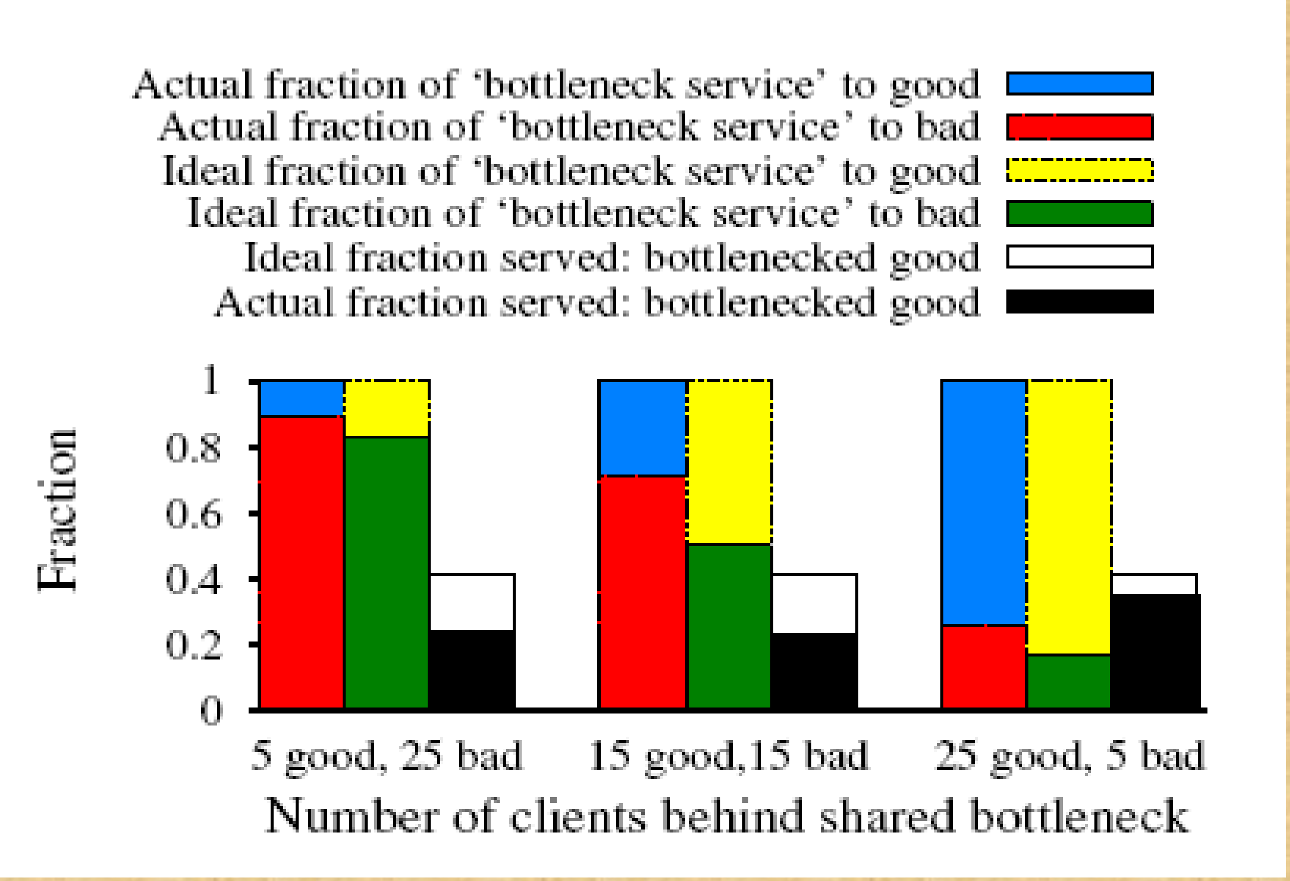
Heterogeneous Network Conditions

- Good clients with long RTTs do worse than any bad clients
- "Effect is limited"
  - No one gets > 2*ideal
  - No one gets < 1/2*ideal

Good and Bad Sharing a Bottleneck

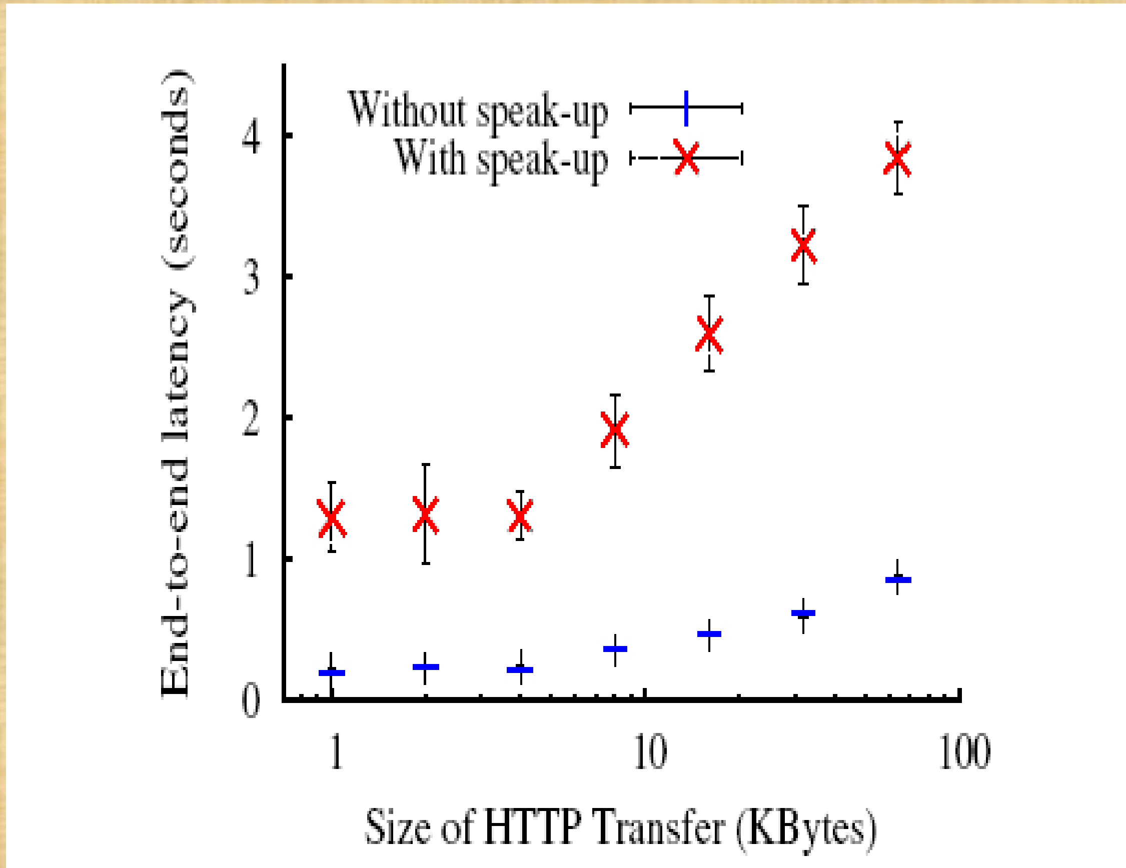


Good and Bad Sharing a Bottleneck



$$f_{id} = \frac{\frac{G_l}{G+B}c}{g_l} = \frac{\frac{G_l}{40}30}{g_l} = \frac{2\frac{20}{60}n30}{40n\lambda} = \frac{2\frac{20}{60}n30}{80n} = 0.25.$$

Impact of speak-up on other traffic



Setup:
10 good speak-up clients,
2 Mbits/s;
H, a host that runs the HTTP client wget. 2 Mbits/s;
Bottleneck link, *m:* 1 Mbit/s;
one-way delay 100 ms;
the thinner and *S.*

In each experiment,
H downloads a file from S 100 times.

Conclusions

Conclusions

- This article presents the design, implementation, analysis, and experimental evaluation of *speak-up,* a defense against *application-level distributed denial-of-service (DDoS).*
- With speak-up, a victimized server encourages all clients, resources permitting, *to automatically send higher volumes of traffic.*

Conclusions

- Advantages
  - Network elements don’t need to change.
  - Only need to modify servers and add thinners.
- Disadvantages
  - Everyone floods, so harder to detect bad clients.
  - Hurts edge networks.
  - Rendered useless if access links to thinner are saturated.

Questions?