DDoS Defense by Offense

Michael Walfish, Mythili Vutukuru,
Hari Balakrishnan, David Karger, and Scott Shenker*

MIT Computer Science and AI Lab

** UC Berkeley and ICSI*

14 September 2006

Today's DDoS Attackers and Defenders

- The modern DDoS attacker
  - Strong motives → attacks evolving
  - Tries to make its traffic look legitimate
- The modern DDoS defender
  - Ethos: "detect, then deny"
- The post-modern DDoS defender (us)
  - No attempt at reliable differentiation:

When a server is under attack, encourage *all* clients to send *more* traffic to the server

I. Justification

II. Realization

III. Discussion

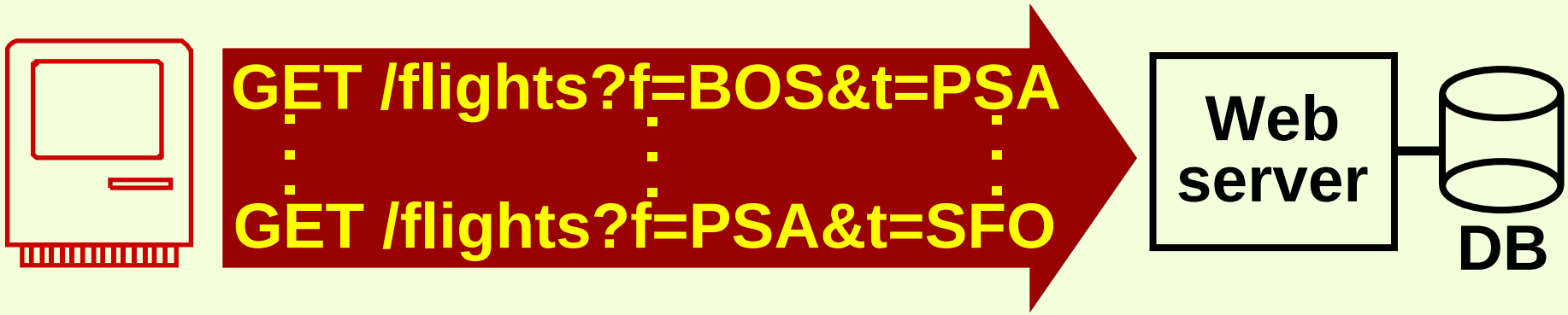
I. Justification: Where? Why?

II. Realization

III. Discussion

Application-level Attacks

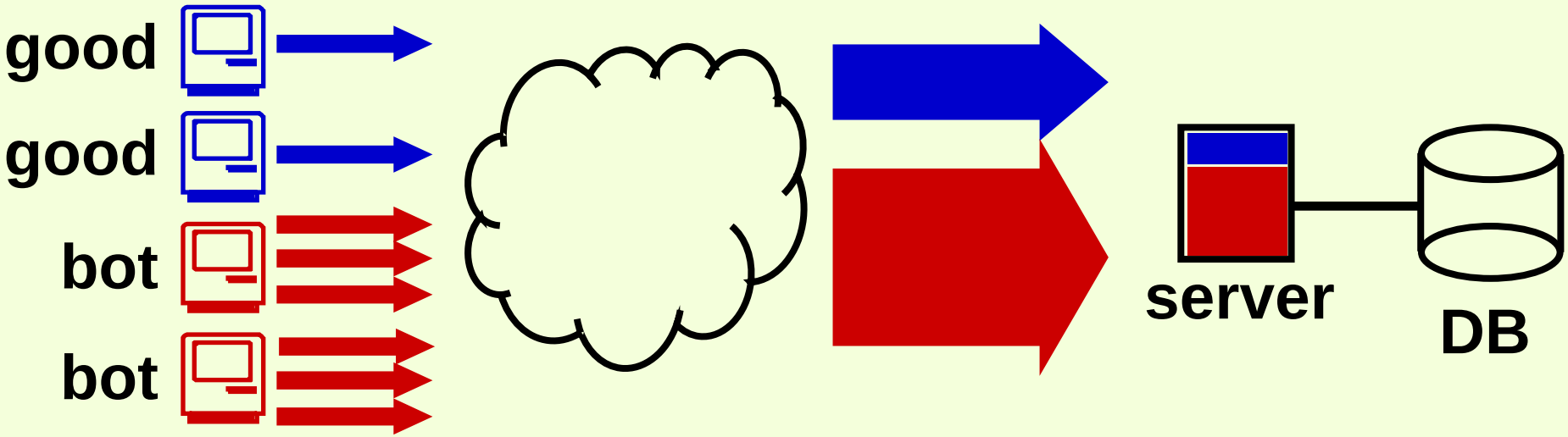
- *Bots* send requests that look legitimate
  - Overloads resource like CPU, disk (not link)



Key Challenge:

- Can't tell request was issued with ill intent
  - Clientele may be unknown
  - Proof-of-humanity not sufficient

App-Level DDoS on Defenseless Server

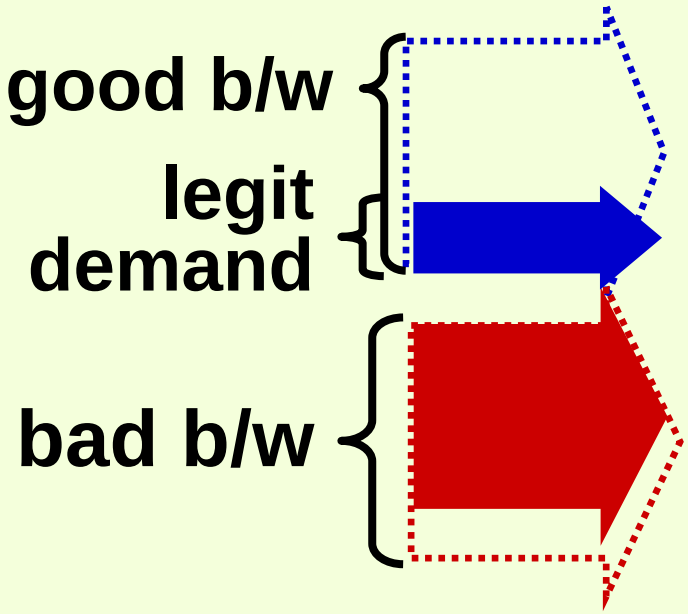


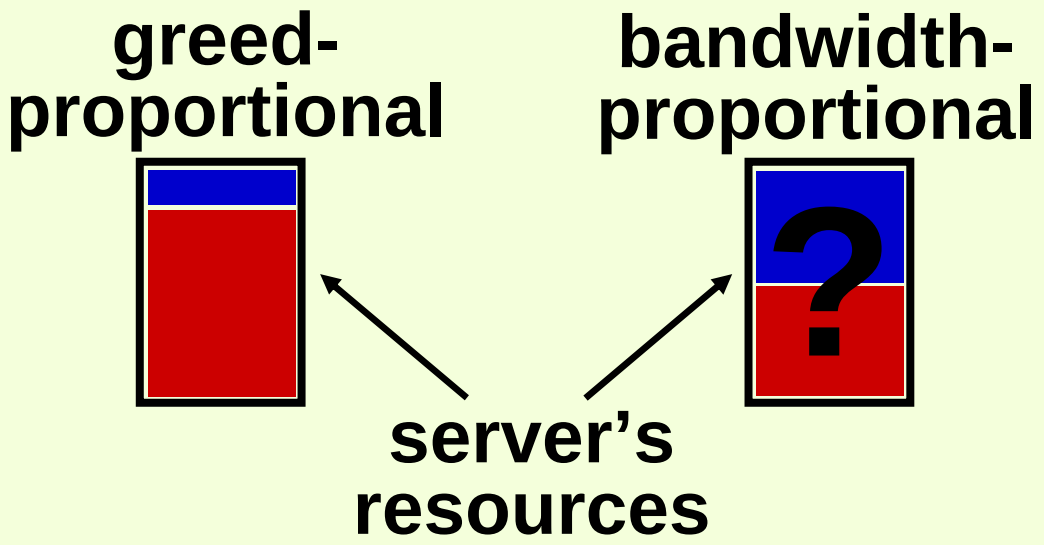
- Server overloaded; drops randomly
- Thus, attackers get the bulk of the server
- This server allocation is *greed-proportional*

Must change the allocation ...

... without differentiating good and bad

Our Goal: Bandwidth-Proportional Alloc.





- Dole out units of service based on client b/w
- Why better than greed-proportional?

  Because good clients have more spare capacity

  (For now, assuming bot b/w ~ good b/w)

What Should the Goal Be?

- Ideal: fair allocation
  - Best possible if you can't detect bad clients
- But this ideal is hard to achieve
  - Proxies
  - IP addr hijacking and harvesting (bots *reachable* at stolen IP addresses)
- Settle for approximately fair allocation

Why Choose Bandwidth-Proportional?

- Clients can’t fake b/w and b/w is measurable
  - Provided clients are forced to consume it

When a server is under attack, encourage *all* clients to send *more* traffic to the server

Why Choose Bandwidth-Proportional?

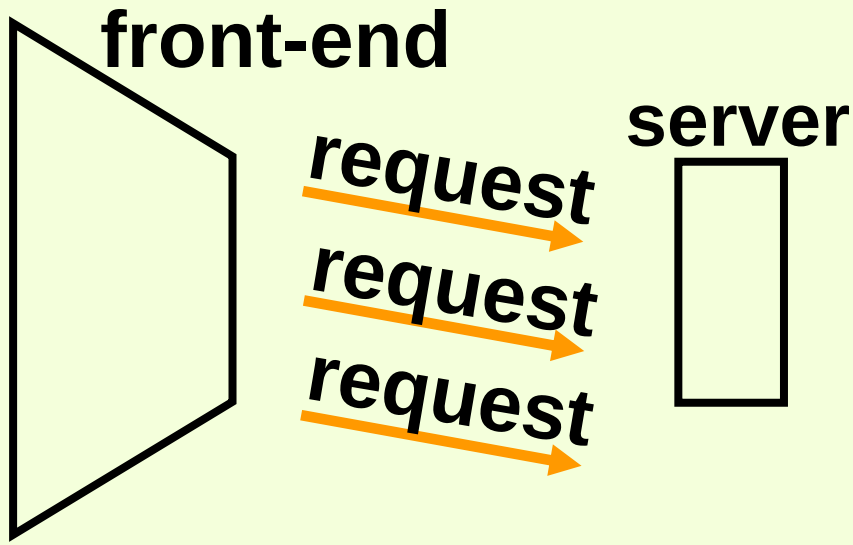
- Clients can't fake b/w and b/w is measurable
  - Provided clients are forced to consume it
  - "Taxation without identification"
- CPU also a possibility (proof of work)
  - Though harder to set the price ...
  - ... and pegging link better than pegging CPU
- How to achieve? With our system, *speak-up*

I. Justification: Where? Why?

II. Realization:
{Design, Impl, Eval} of *Speak-up*

III. Discussion

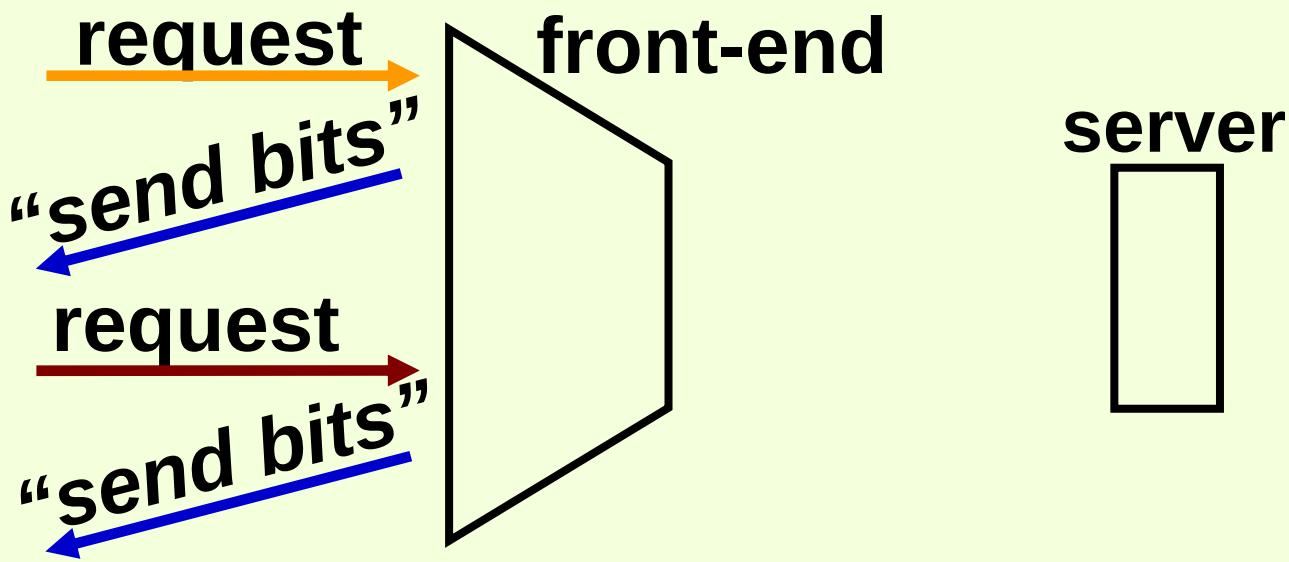
Speak-up in a Nutshell



Only under server overload:

- Front-end admits requests periodically

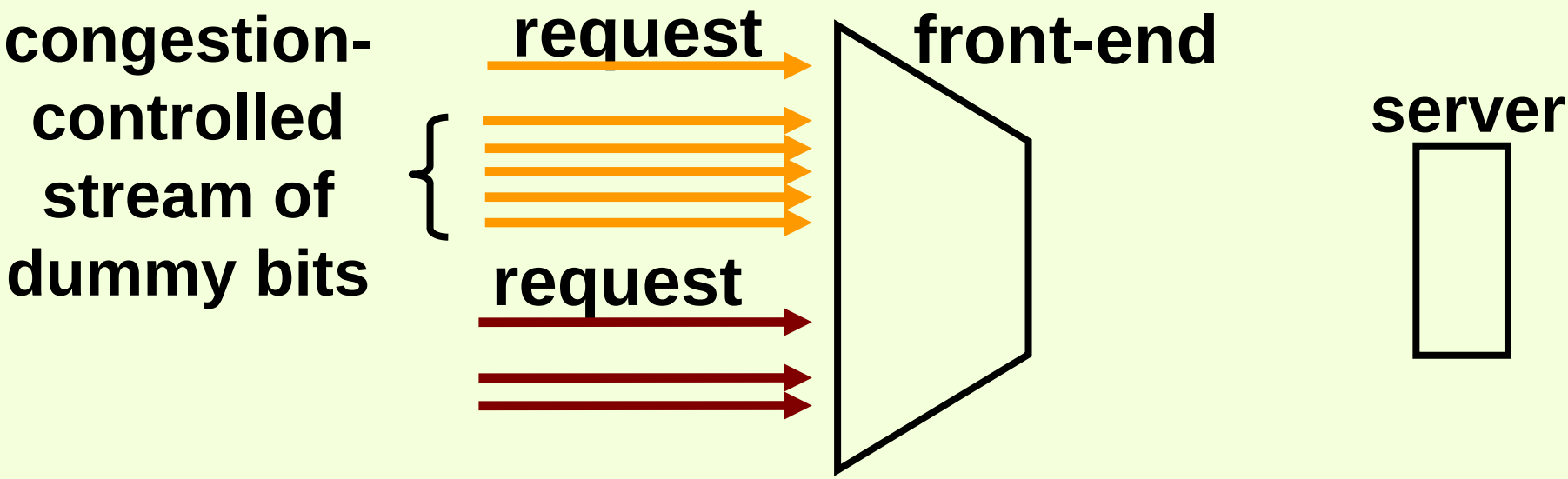
Speak-up in a Nutshell



Only under server overload:

- Front-end admits requests periodically

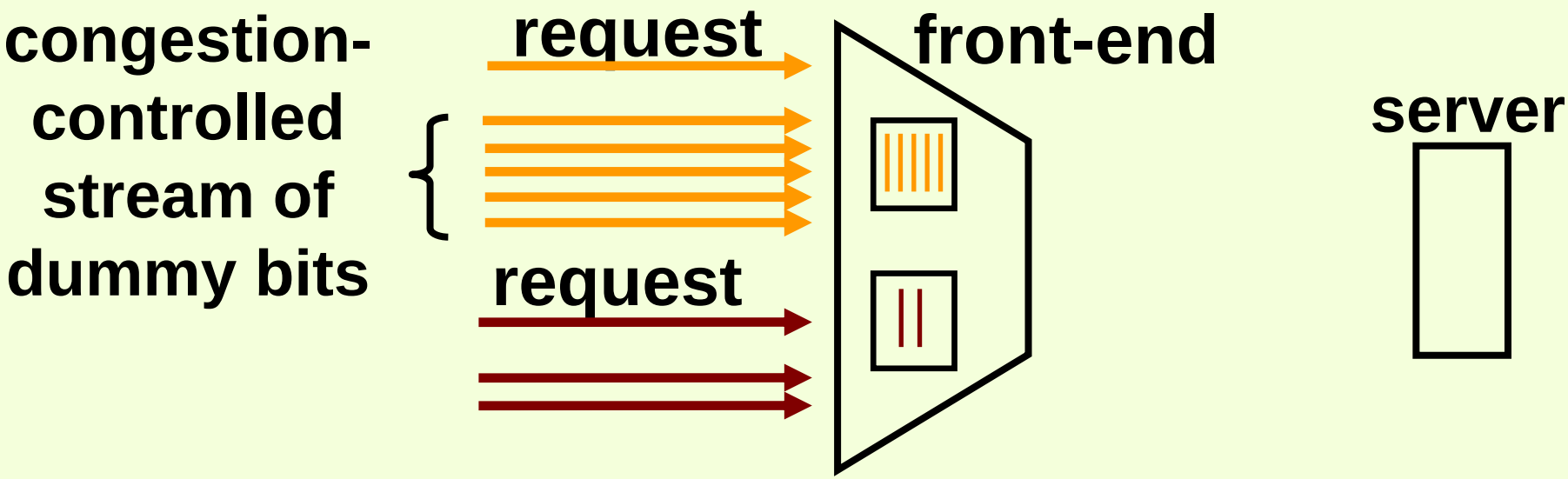
Speak-up in a Nutshell



Only under server overload:

- Front-end admits requests periodically

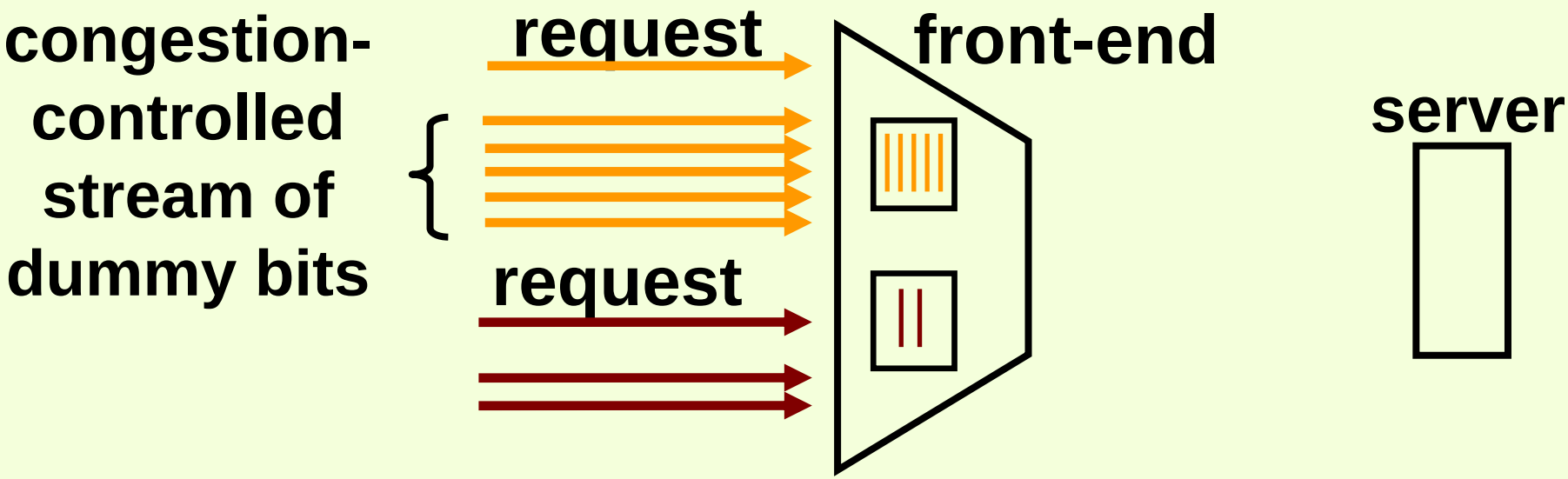
Speak-up in a Nutshell



Only under server overload:

- Front-end admits requests periodically

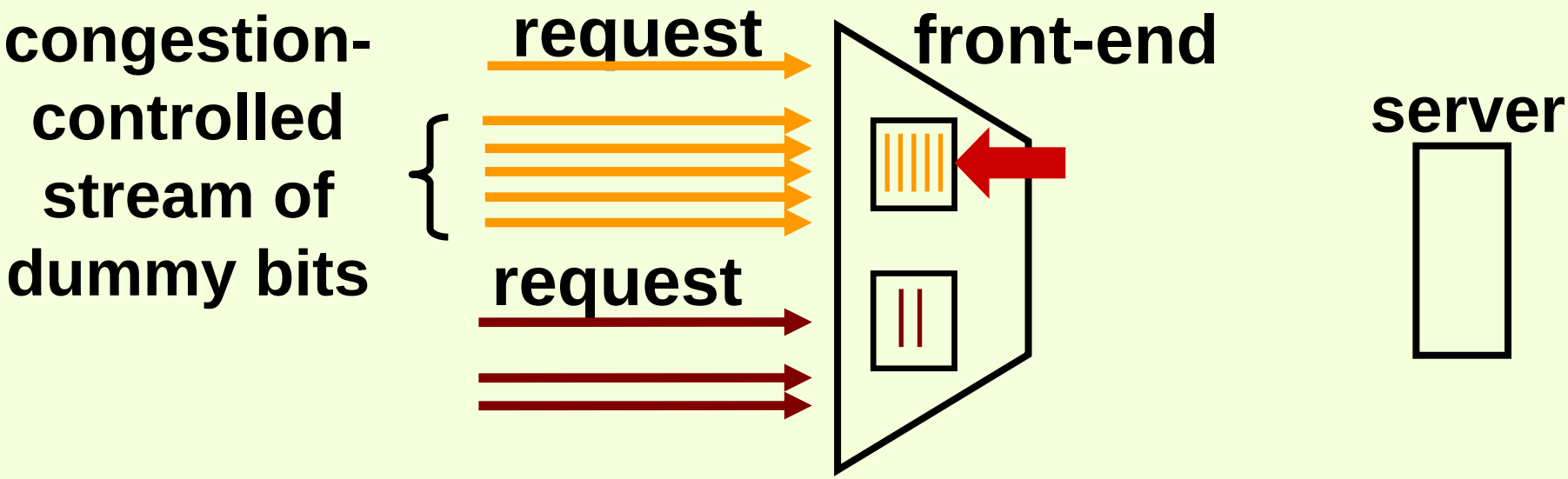
Speak-up in a Nutshell



Only under server overload:

- Front-end admits requests periodically
- Which request to admit?

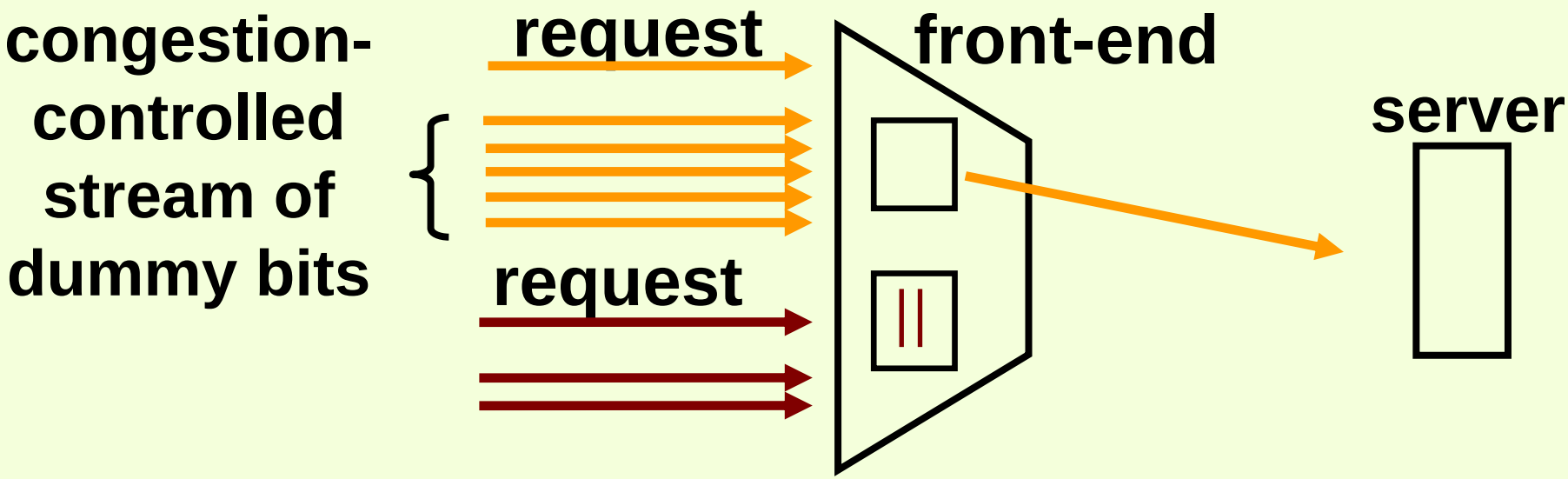
Speak-up in a Nutshell



Only under server overload:

- Front-end admits requests periodically
- Which request to admit? “Highest” sender

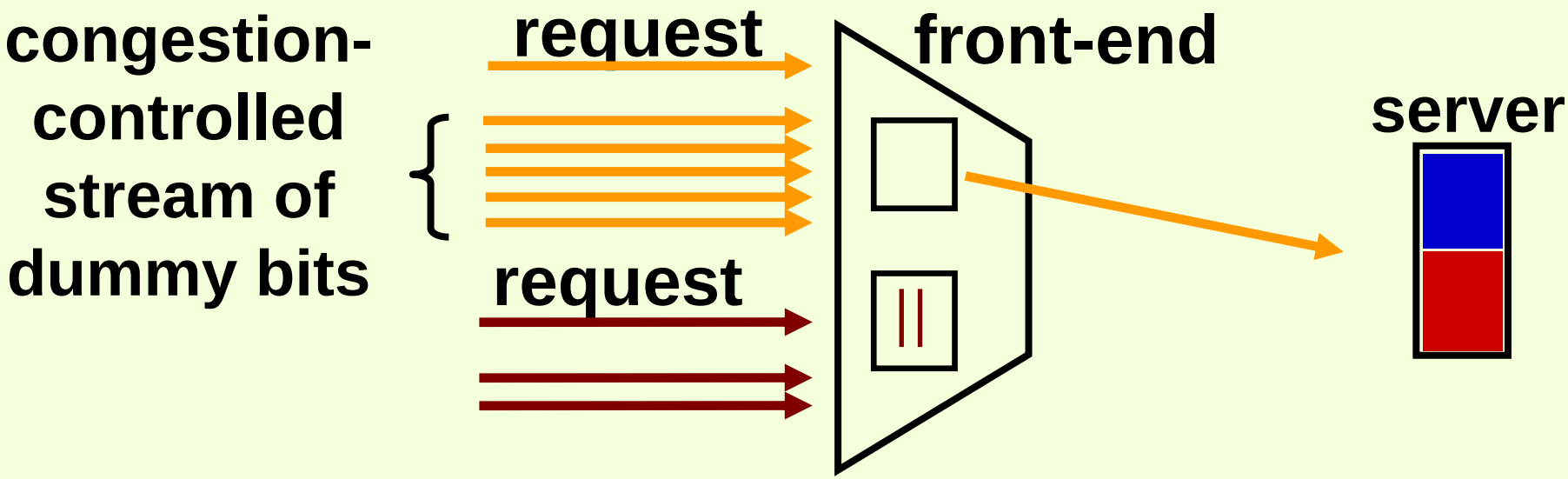
Speak-up in a Nutshell



Only under server overload:

- Front-end admits requests periodically
- Which request to admit? "Highest" sender
- Others keep sending and *eventually win*

Speak-up in a Nutshell

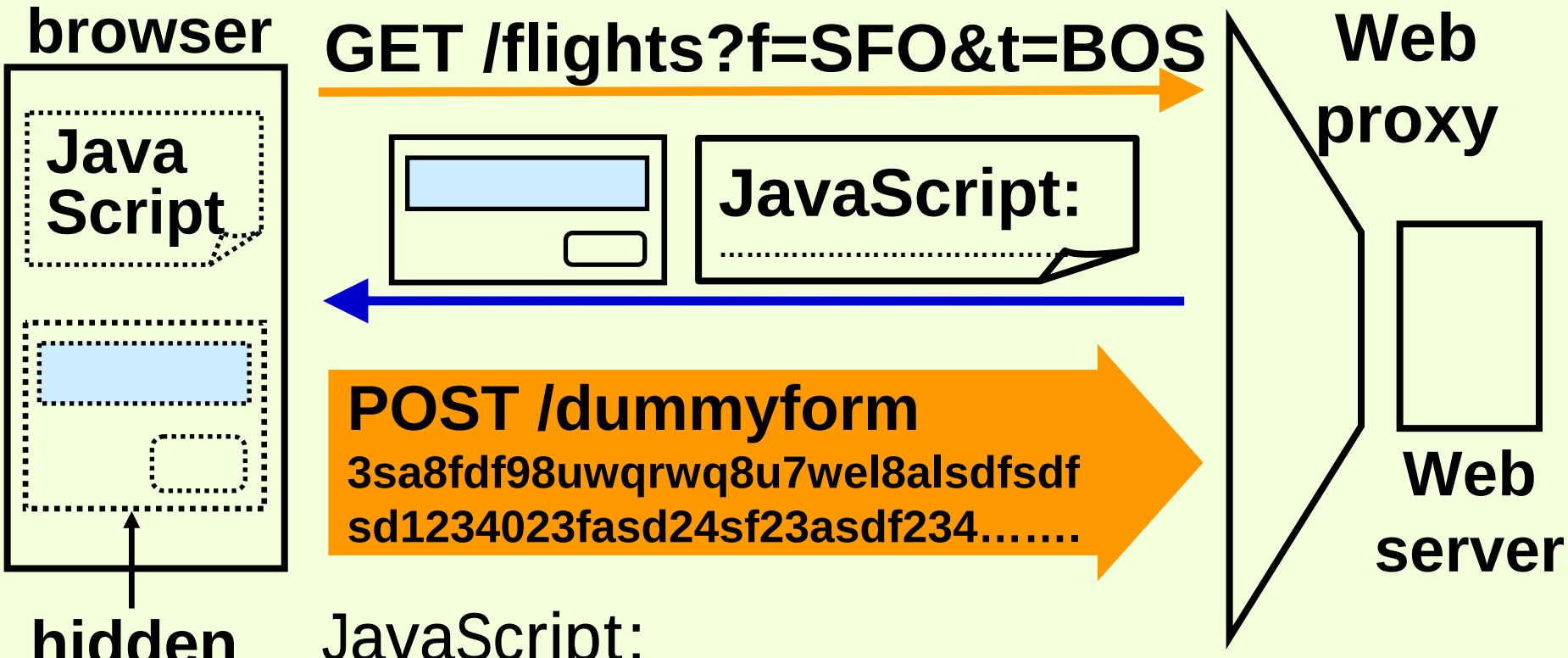


Only under server overload:

- Front-end admits requests periodically
- Which request to admit? "Highest" sender
- Others keep sending and *eventually win*

(Allocation prop. to b/w: proved in paper.)

Implementation (Needs No Client Changes)

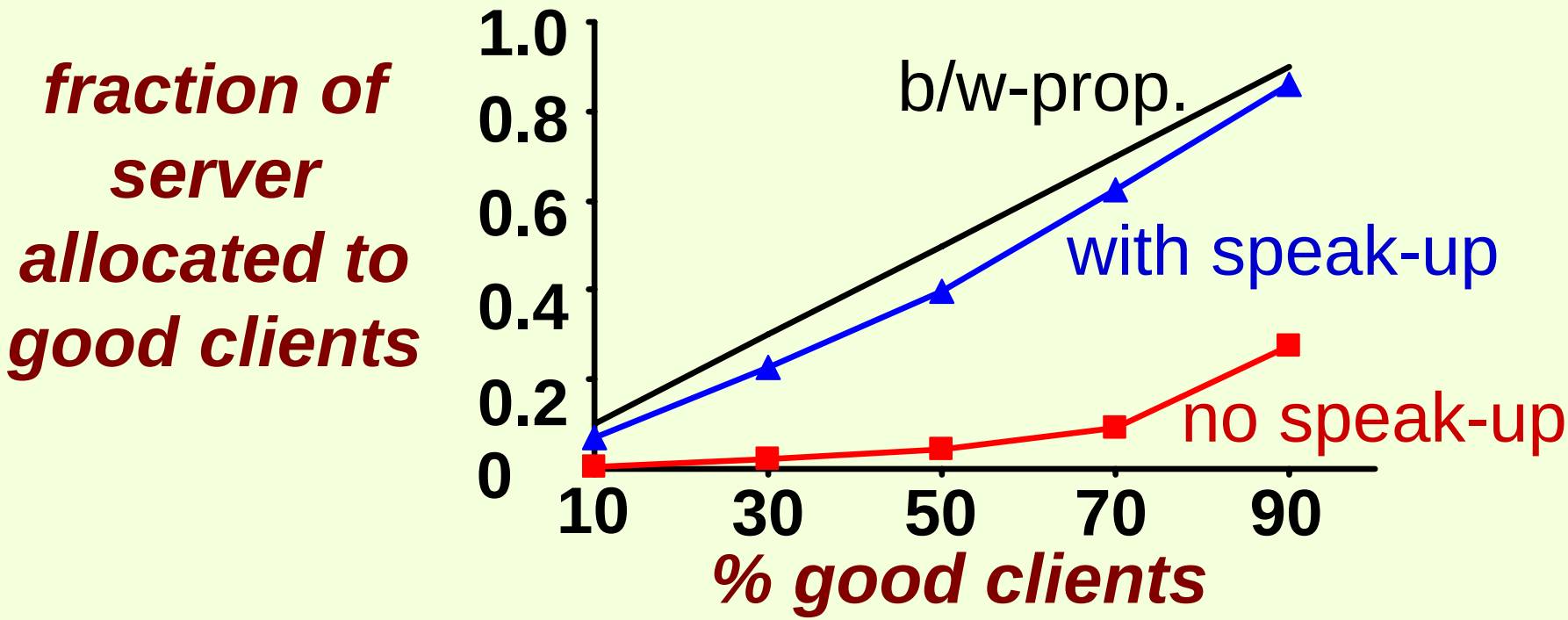


JavaScript:

- Client constructs 1MByte string
- POSTs string in form

Proxy: ends POST after client wins

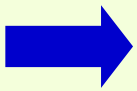
The Implementation Roughly Meets Its Goal



- 50 clients; all have 2 Mbits/s bandwidth
- Vary number of good and bad
- Good clients: 2 reqs/s; bad clients: 40 reqs/s
- Server capacity: 100 reqs/s

I. Justification: Where? Why?

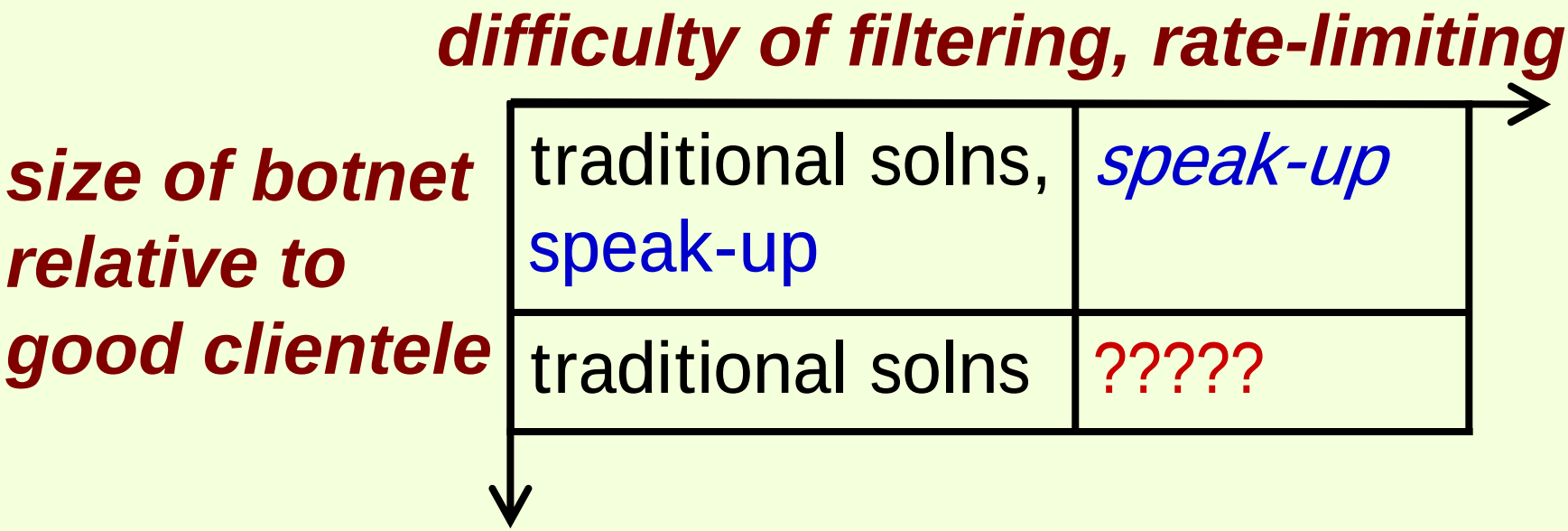
II. Realization: Design, Impl, Eval



III. Discussion: Applicability, Objections, Related Work, Summary

Conditions That Call for Speak-up

1. Application-level attack
2. Hard to filter, hard to rate-limit explicitly
3. Botnet not much larger than good clientele
4. Front-end has a lot of bandwidth



How Often do the Conditions Hold?

Hard to know definitively, but:

- Attacks moving toward application-level
- Proxies widespread; IP addr stealing happens
- Botnet size vs good clientele size:
  - Many less than 10k or even smaller [Symantec, Rajab et al. IMC06, Arbor, LADS, McCarty IEEE SecPriv03]
  - Anecdotally, botnets getting smaller
  - (Smaller botnets will drive smarter attacks)
- Many sites have access to a lot of bandwidth

Some Objections to Speak-up

- Won't it harm the network?
  - Inflation only in traffic to attacked sites
- Clients have unequal bandwidth
  - True: speak-up is only roughly fair
  - Possible solution using proxies
- Many others (see paper)

Other Defenses to App-Level DDoS

- Detect and block attackers
  - CAPTCHAs [Morein et al. CCS03, Gligor IWSP03, Kandula et al. NSDI05]
  - Profiling [Mazu, Arbor, Ranjan et al. INFOCOM06, etc.]
- Rate-limiting [Fair Queuing, Banga et al. OSDI99, Kandula et al. NSDI05]
- Proof-of-work [Dwork & Naor 92, Juels & Brainard NDSS99, Aura et al. IWSP00, Mankins et al. ACSAC01, Wang & Reiter Oakland03, Hashcash, etc.]
- "Dilute" attackers (make clients repeat requests) [Gunter et al. NDSS04, Sherr et al. WSNP05]

Summary and Take-home Points

- DDoS evolving → traditional methods (detection, rate-limiting) less effective
- Taxation fairer than explicit identification
- For app-level attacks, we propose speak-up
  - Allocates server according to client b/w
- Speak-up trades b/w for server computation