Client Puzzles

Defending Against Denial-of-Service Attacks with Puzzle Auctions

Outline

- Motivation
- Auction Protocol
- TCP Puzzle Auction
- TCP Client Puzzle
- Implementation
- Experiment Results
- Questions

Defending Against Denial-of-Service Attacks with Puzzle Auctions

[Wang & Reiter, IEEE Symposium on Security and Privacy '03]

- Clients choose puzzle difficulty
- Whoever solves hardest puzzle, gets server resources

Auction Protocol Motivation

- Determining if a server is under attack is difficult
- Clients determine whether server is under attack (based on request fulfillment)
- Clients don’t have to do work unless server is under attack
- Adversaries resources unknown

Auction Protocol

- Why client chooses difficulty?
  - Client and adversary resources unknown
  - Relative amount of resources yields the puzzle difficulty
  - Adversary can only do so much damage (maximum amount of work to do minimum damage)
- How do bids interfere with future resources?

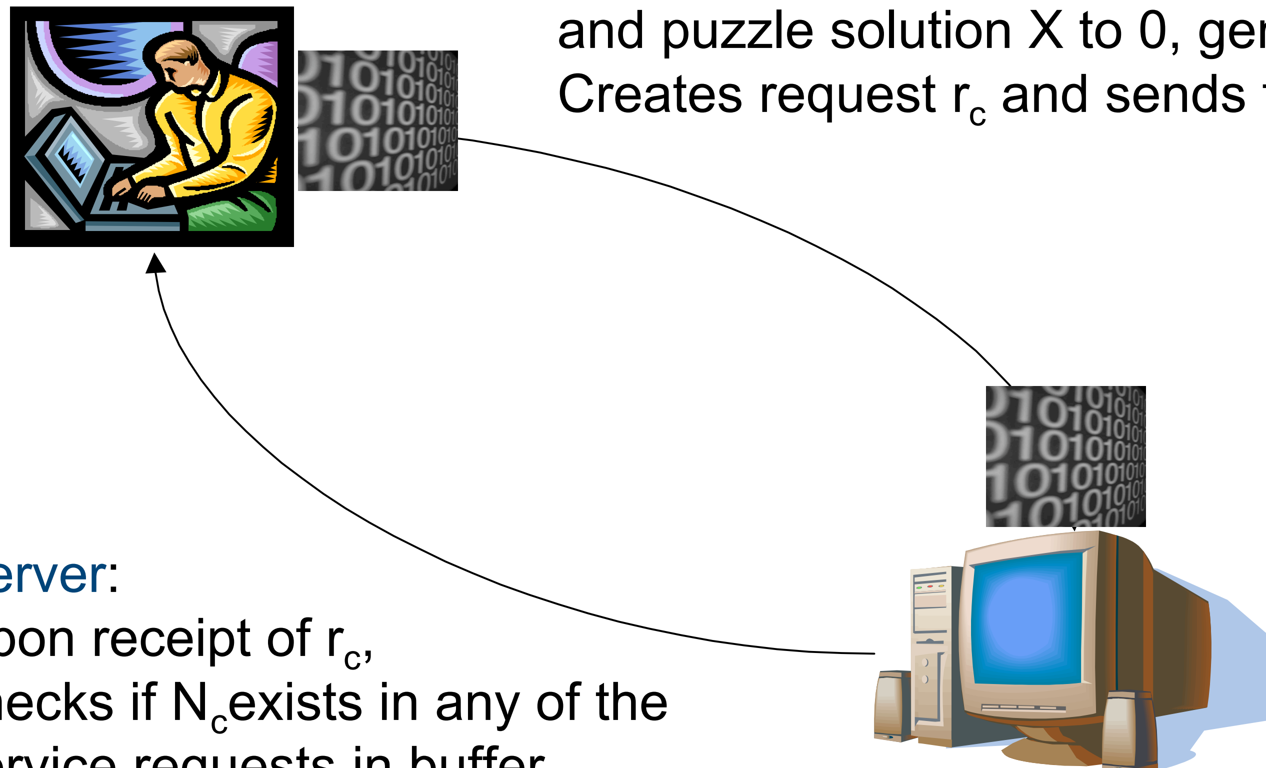
Biggest Challenge: Deployment

- Legitimate clients have to implement this system for this to be used
- Without legitimate servers, legitimate clients won't install it
- If servers install it first, adversaries can take advantage of it

Auction Protocol

Client

Client:
Sets target puzzle difficulty to 0
and puzzle solution X to 0, generates N_c
Creates request r_c and sends to Server



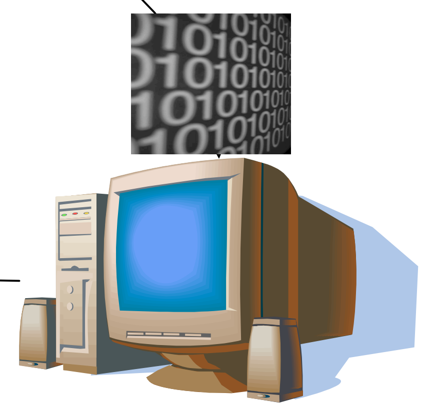
Server:
Upon receipt of r_c,
checks if N_c exists in any of the
service requests in buffer,
if so sends service failure to client,
with current server nonce N_s

Server

Auction Protocol

Client

Server:
Checks buffer queue of service requests. If it's not full, adds r_c to buffer queue

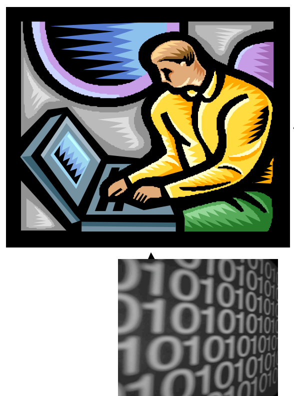


Server

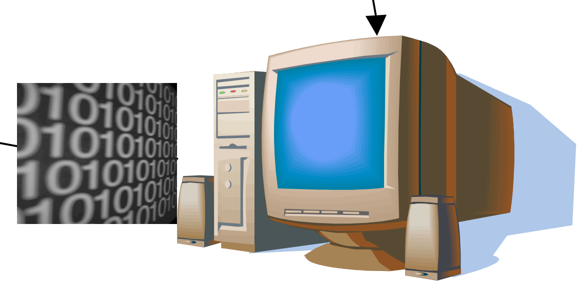
Server:
1) Checks puzzle difficulty of existing service requests in buffer
2) If there is a difficulty lower than r_c's, drop that request and add r_c
3) Otherwise, send notification of service failure with server nonce N_s

Auction Protocol

Client



Server:
Periodically checks buffer queue
for completed requests
and clears them



Server

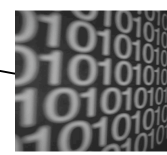
Client:
Brute force searches puzzle
solution, until puzzle difficulty
is either greater than the
target puzzle difficulty or its
maximum number of hash operations

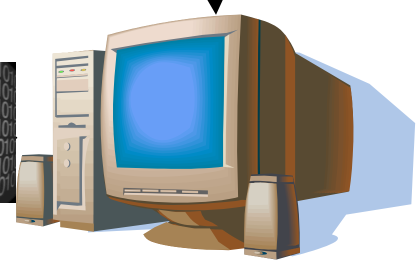
Auction Protocol

Client

Client:
Upon notification of service failure, extracts N_s and increases its bid

Client:
Brute force searches puzzle solution, until puzzle difficulty is either greater than the target puzzle difficulty or its maximum number of hash operations





Server

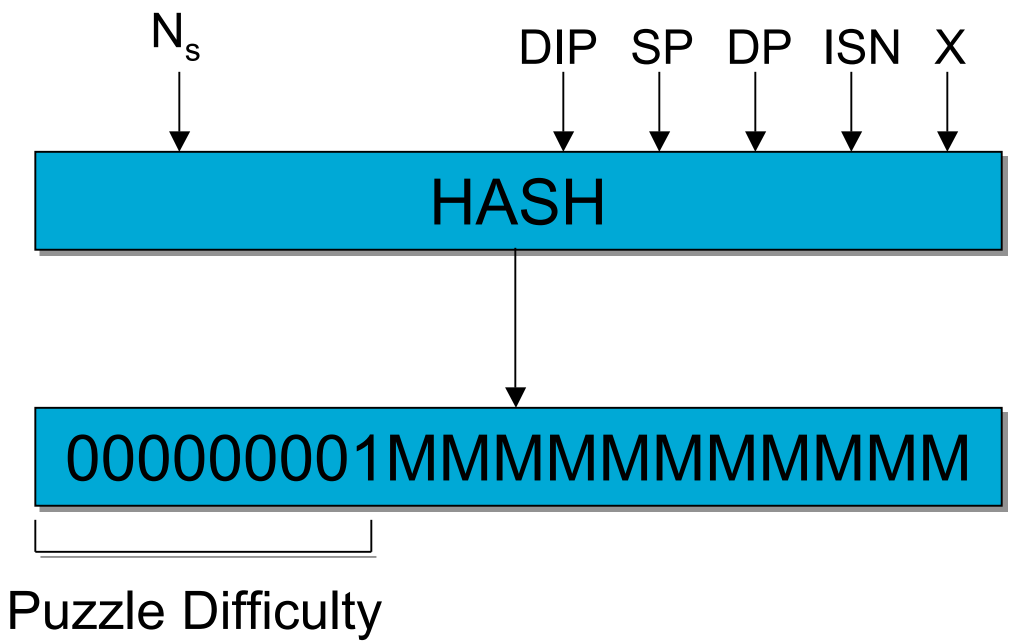
TCP Puzzle Auction

- Defends against connection-depletion attacks on TCP
- Negligible overhead to server
- Interoperable with clients that have unmodified kernels

TCP Client Puzzle

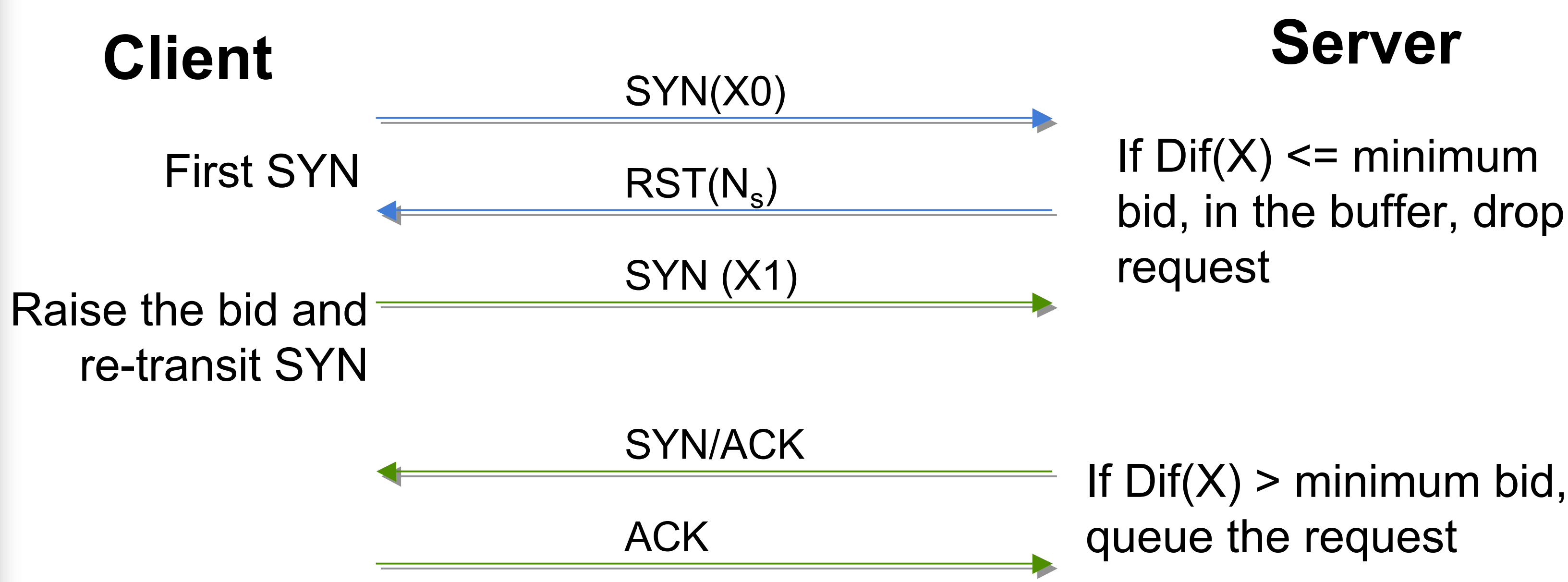
- X: Puzzle solution
- N_c: source IP address (SIP), destination IP address (DIP), source port (SP), destination port (DP), initial sequence number (ISN)
- N_s: hash function with client IP address and server secret as input
  - Changes after each nonce period
  - Server secret increases for each nonce period

TCP Client Puzzle



Replace first x bits of hash with 0 to modify difficulty

TCP Puzzle Auction



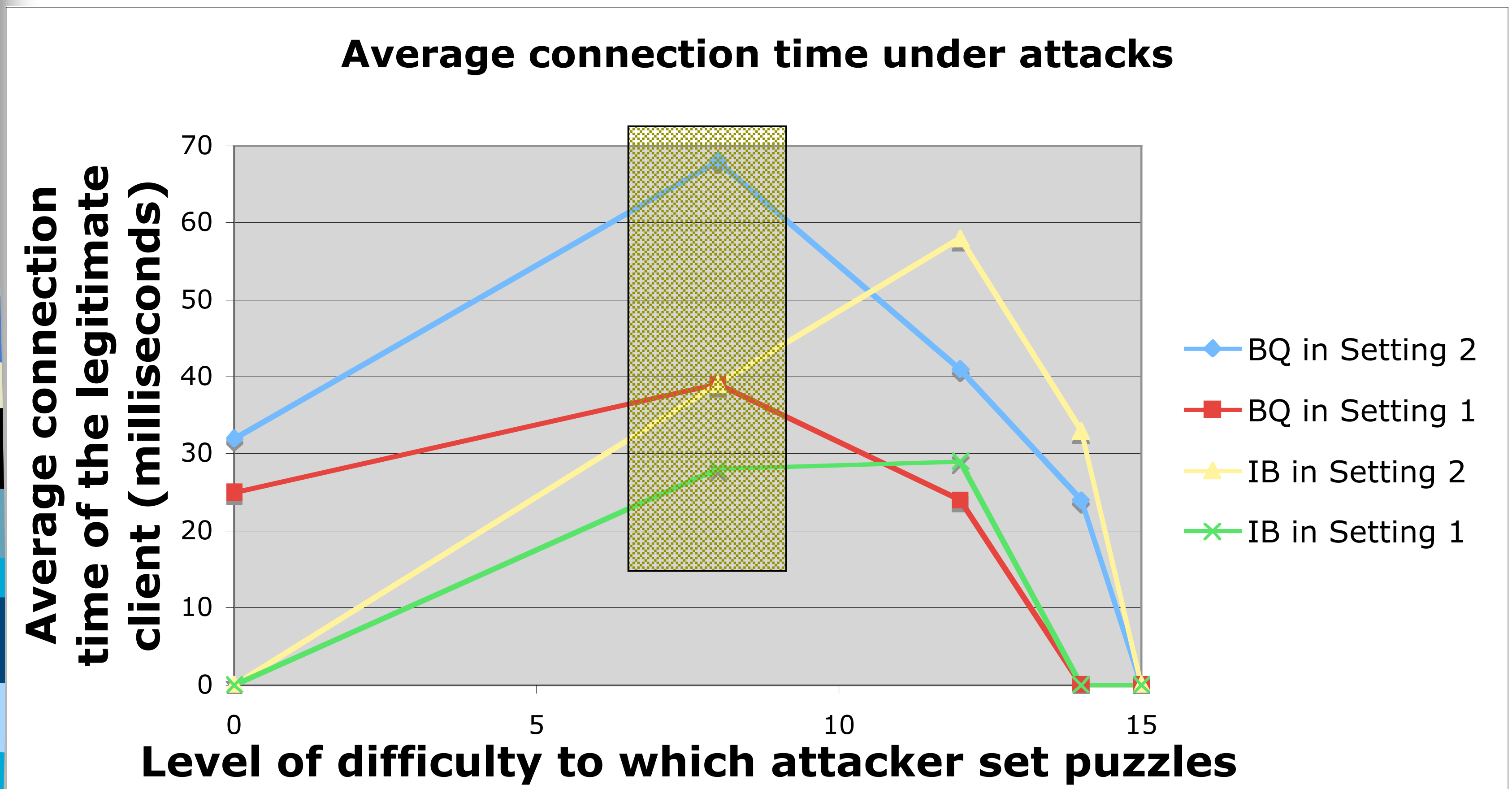
Implementation

- Client
  - Pentium Pro 199 Mhz machine with 64MB memory
- Server
  - Intel PIII/600 with 256MB memory
- Attacker
  - Two Intel PIII/1GHz CPUs and 1GB memory
- All have 2.4.17 Linux kernel
- On 100Mbps campus network

Experiment Results

- Study 1: Puzzle overhead
  - Connection time of 255.4 μs vs. 250.8 μs
- Study 2: System Performance
  - Two server settings
    - 9 seconds to discard half-open connections (Setting 2)
    - 3 seconds to discard half-open connections (Setting 1)
  - Two strategies
    - Bid & Query (BQ)
    - Incremental Bidding (IB)

Server Performance



Analysis of Results

- IB & BQ so close
- Why does this happen?
- What does this mean?

Summary (Technical Contributions)

- Applies auction protocol to client puzzles
- Compatible with unmodified kernels
- Server does not have to determine when it is under attack
- Evens playing field between legitimate clients and adversaries

Waters, et al. paper

- Questions
- Critique

Client Puzzle Reuse

- Client can tailor puzzles to a specific server
- Each puzzle can be “re-used” at different servers
- Adversary can take advantage of this side effect

Bastion

- Bastion is integral to this scheme
- No analysis of bastion in the author's implementation
  - How secure is the bastion?
  - Will this scheme work if the bastion is compromised?

Offline computation

- How does client know which servers it will access a priori?
- Is it possible to modify the scheme so that offline computation is practical?

Calculating T

- Paper sets T at 20 mins.
  - Client may have to wait 20 mins. at startup
  - Is this practical?
- Why not decrease T?

Calculating T

- Empirical Results: Finding 100, 20 bit partial collisions

CPU Speed	Memory Size	HashCash (in seconds)
398.252MHz	128MB	269.904
1.6GHz	256MB	149.962
3.2GHz	1GB	36.818
2GHz	3GB	69.290
797MHz	512MB	47.544

- Brute force on the slowest machine was 260s vs. 20 mins. wait time

Figure 1 - 100% CPU?

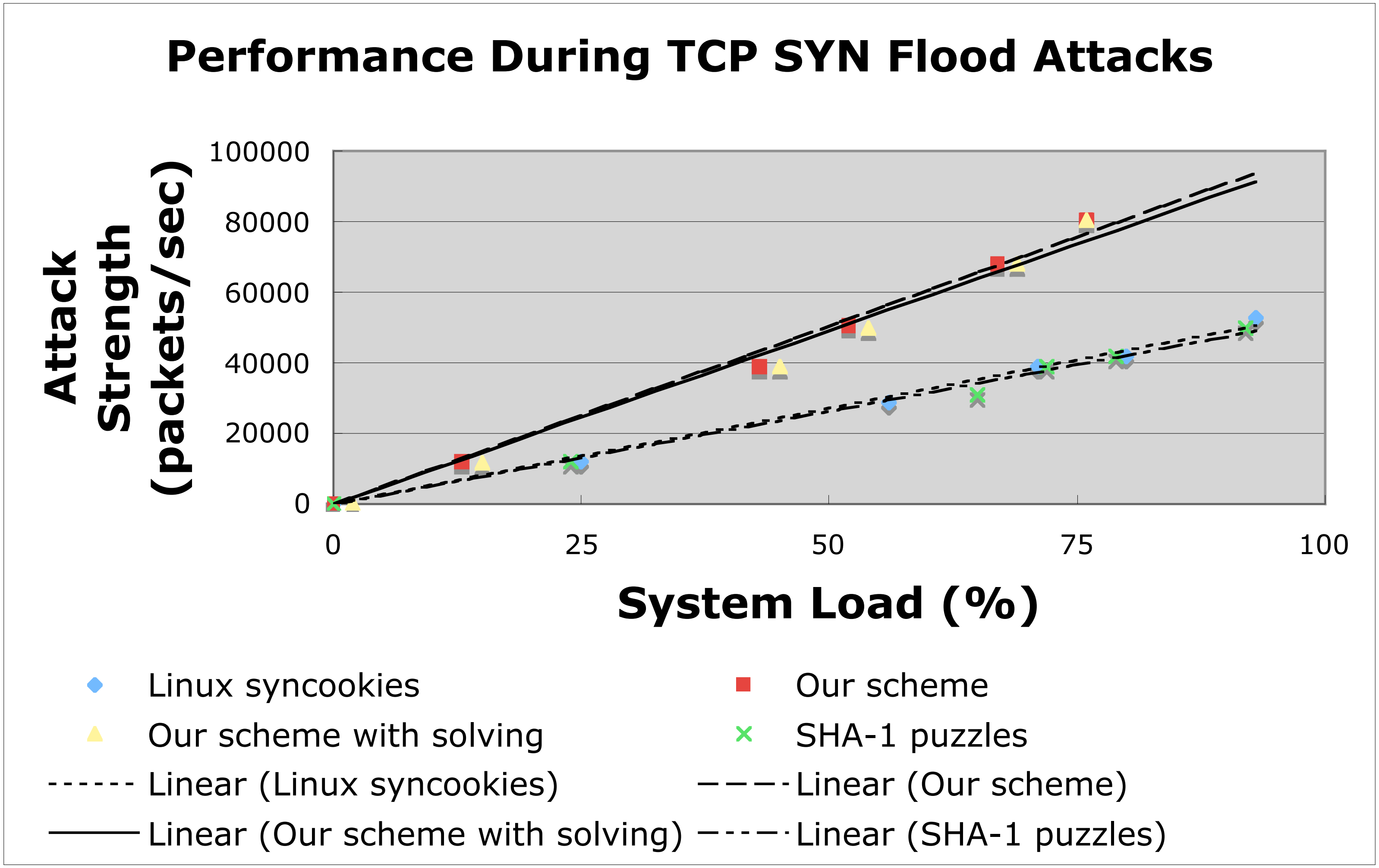
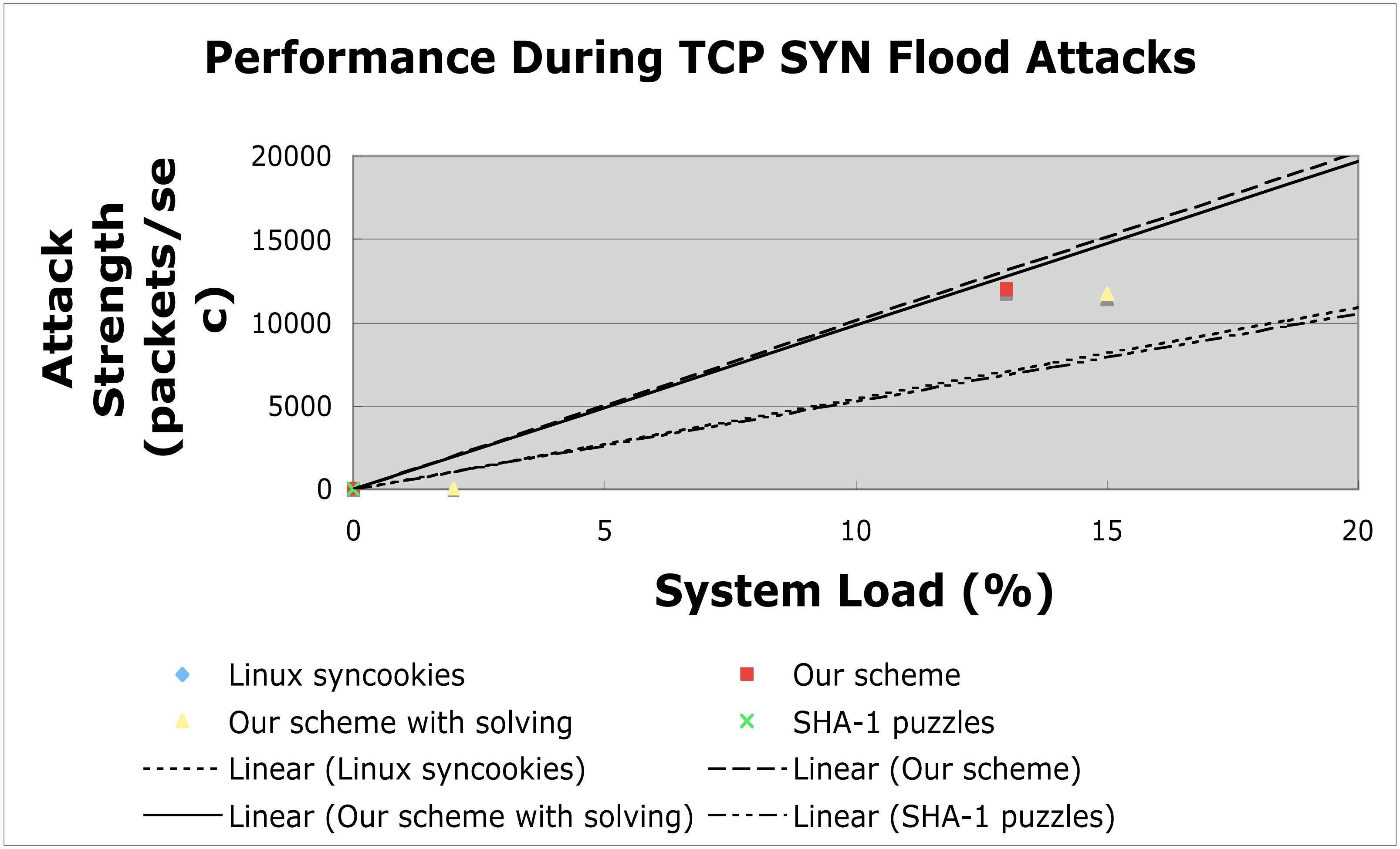


Figure 1 Modified



Analysis & Assumptions

- Channels not varied at all
- Computing advances will benefit clients
  - Doesn't it benefit adversaries also?

Assumptions

- Adversary has 50 zombie machines
  - “Know your Enemy: Tracking Botnets”
    http://www.honeynet.org/papers/bots/
  - Tracked 100 botnets over 4 months
  - **226,585** unique IP addresses joining at least one of the channels
  - Some large botnets up to **50,000** hosts

Additional comments/questions?