Protecting Web Servers from Distributed Denial of Service Attacks

Frank Kargl, Joern Maier, Michael Weber

WWW10, May 1-5, 2001, Hong Kong
ACM 1-58113-348-0/01/0005

Presented by Joe Frate

Goals of Paper

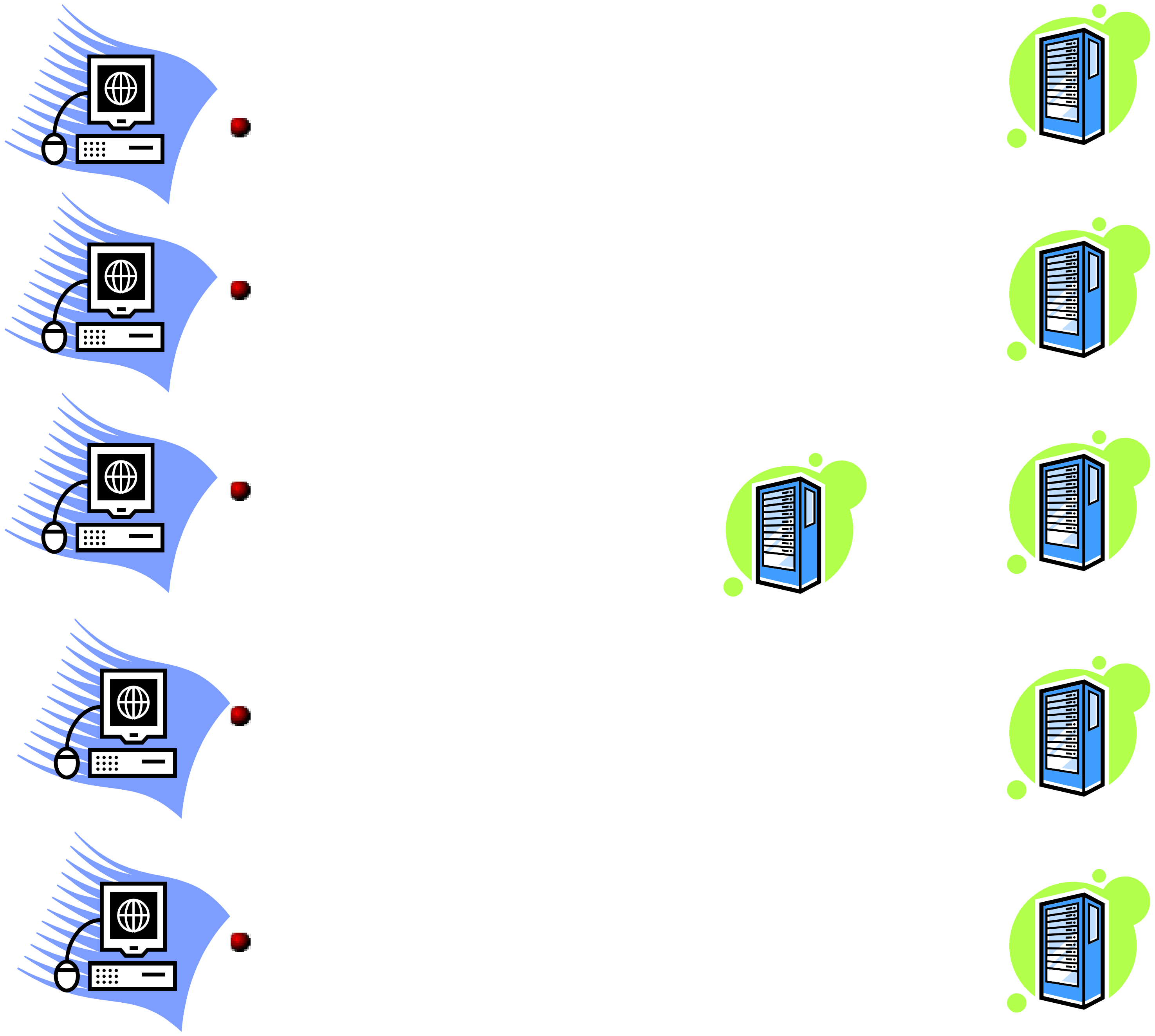
- Distributed Denial of Service Attacks (DDoS)
  - Categorize different forms of attacks
  - Overview of some DDoS tools
- Defense based on Class Based Routing
  - Defend clusters of Web servers
  - Still allow normal traffic during attack ("automatic traffic shaping")
- Performance tests of presented solution

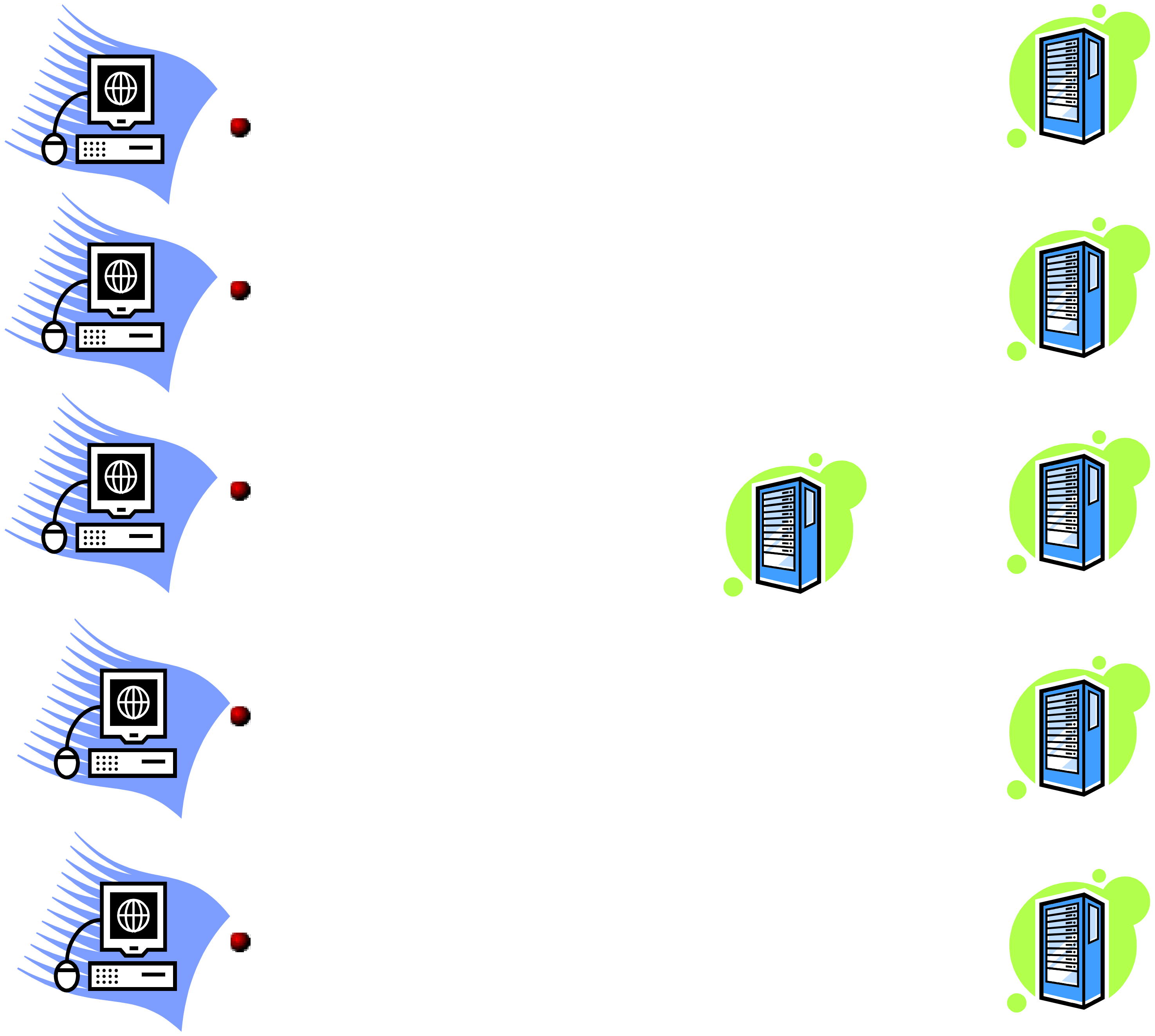
Denial of Service (DoS)

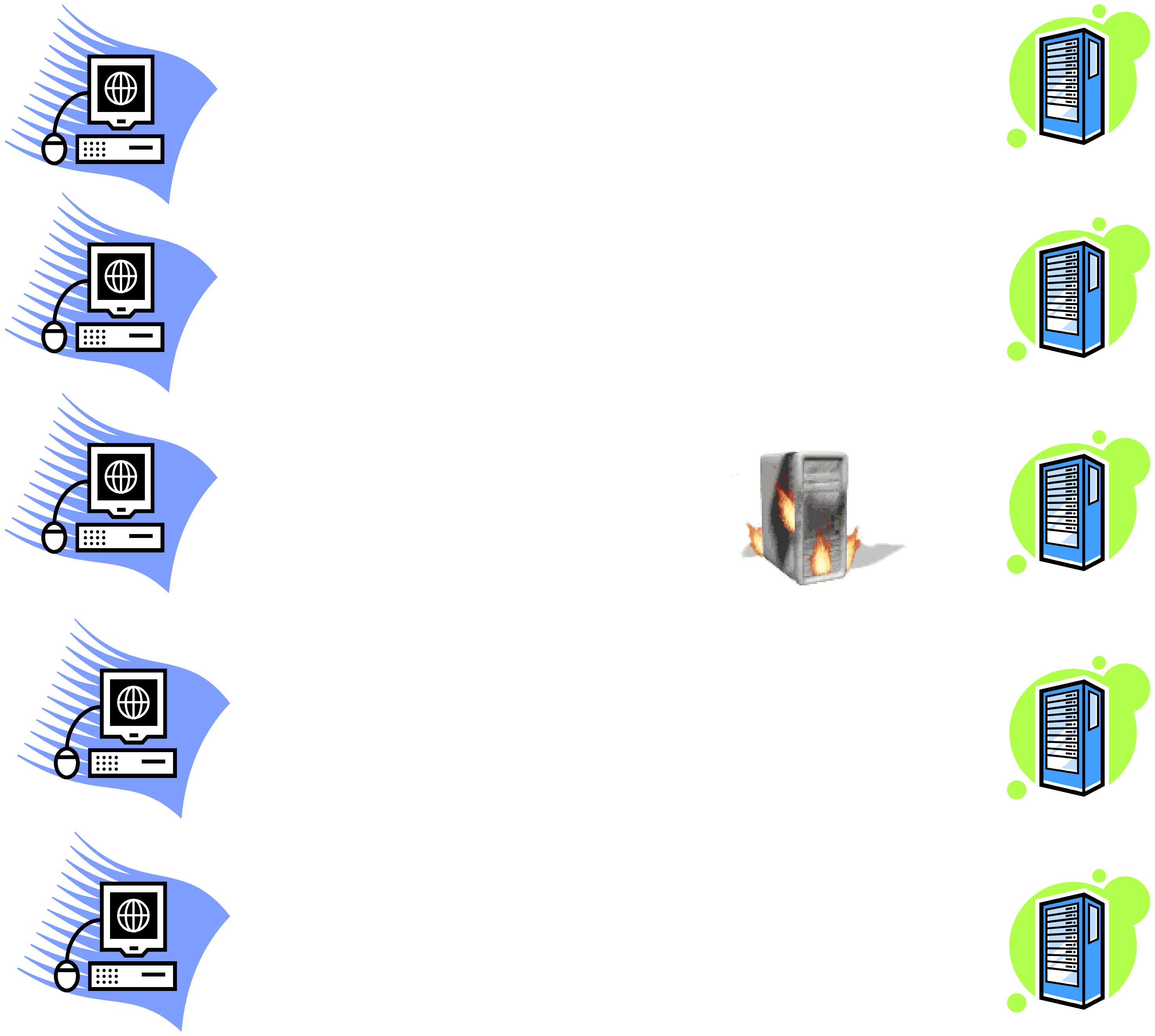
- "… an attack designed to render a computer or network incapable of providing normal services" (WWW Security FAQ)
- To be an "attack", must be intentional

Distributed Denial of Service (DDoS)

- Carried out via networks
- Many computers against target(s)
- Hosts as unwitting accomplices
- Master program initiates attack
- Bandwidth attack: use all available network resources
- Connectivity attack: consume target’s resources







DoS and DDoS Attack Classification

- System attacked
  - Router connecting web server to ISP (cut off Internet access of service)
  - Firewall system (bottleneck so good target)
  - Load balancer (another bottleneck)
  - Web servers (many so higher effort)
  - Services, e.g. database (behind firewall so difficult)

DoS and DDoS Attack Classification (continued)

- Part of system attacked
  - Ethernet link
  - Operating system of a host or router
  - TCP/IP stack of a host or router
  - Application services (database, etc.)
  - Hardware (network card, CPU)

DoS and DDoS Attack Classification (continued)

- Bug exploitation
  - More severe in effects
  - Fix quickly provided by vendors
  - Diligent administration necessary
- Overload
  - Components function according to specs
  - DDoS attack overloads beyond specs
  - Harder to cope with

Example: Cisco 7xxx routers

- IOS/700 Software Version 4.1(1) or 4.1(2)
- Bug: long password in telnet → buffer overflow
- Subsequent reboot
- Attacked system is a router
- Part of system under attack is OS
- Exploits bug

Example: Jolt2

- Attack targeting Microsoft Windows OS
- Continuous stream of ICMP_ECHO_REPLY fragments
- Specially tuned parameters
- CPU and memory usage raises to 100%
- Attack on any resource that uses OS
  - Web servers, Load balancers, Firewalls
- Exploits bug

Example: Microsoft IIS

- Version 4.0 and 5.0 had URL parsing bug
- Inefficient implementation of escape sequence decoding
- Long strings with escape characters effectively stopped web server
- Attack against web server application
- Exploits implementation bug (in IIS)

Example: Smurf attack

- Uses amplifier sites to multiply traffic
- Send ICMP_ECHO_REQUEST packets with spoofed sender address, to one or several subnet broadcast addresses
- Stations on subnet reply to the packets, to spoofed address
- Congestion in target’s network connection
- Attacks web servers (usually), routers, etc.
- No bug exploiting, perhaps faulty config

Example: SYN flood attacks

- Send SYN packets to target
- Do not reply to SYN-ACK packets
- Spoofed source addresses of non-existent or inactive hosts
- Leaves half-open connections that fills queue
- Stops ordinary clients from connecting
- Attacks web servers or load-balancers
- Exploits 3-way handshake in TCP
- Implementation bug? Design flaw?

Example: Apache web servers

- Apache MIME flooding
- Apache Sioux Attack
- Specially formatted HTTP requests could make web server use up huge amounts of memory, taking down server
- Target a specific web server software
- Exploits implementation bug

DDoS attack architecture

- Architecture of DDoS attack tools similar
- DDoS attack carried out by “daemons”
- Daemons controlled by a handler
- Client tools to activate handler
- Handlers and daemons must be installed onto compromised computers
- Intrusion of these computers follows typical pattern

Intrusion to install handlers and daemons

- Stolen account used to setup repository
  - Scanning tools
  - Attack tools (e.g., buffer overrun)
  - Root kits
  - Sniffers
  - DDoS handler and daemon programs
  - Lists of vulnerable and compromised hosts
  - Etc.

Intrusion to install handlers and daemons (continued)

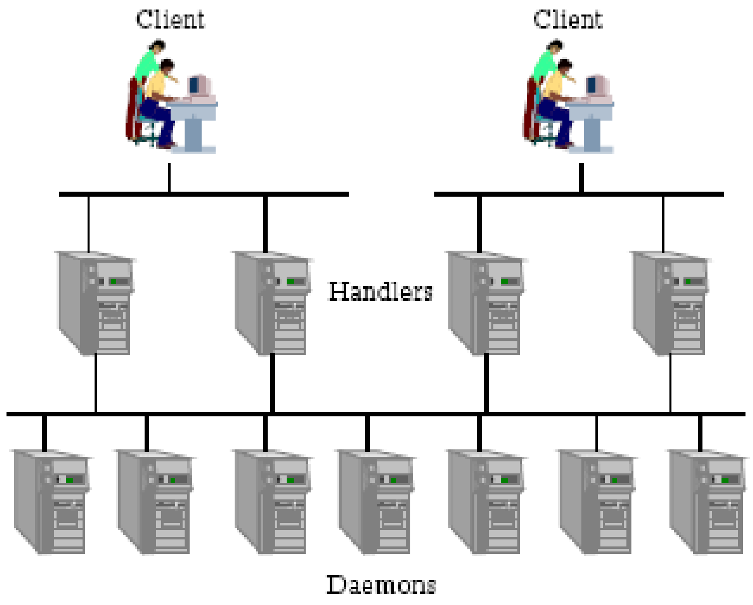
- Scan on large ranges of network blocks, to identify potential targets. Targets would include:
  - Systems running services known to have remotely exploitable buffer overflow security bugs
  - Wu-ftpd, RPC services for cmsd, statd, ttdb-serverd, amd, etc.

Intrusion to install handlers and daemons (continued)

- Use list of vulnerable systems to create script that performs the exploit
  - Set up command shell that runs under root account
  - Script listens on a TCP port
  - Connect to this port to confirm the success of the exploit (i.e., conform that script was successfully installed on the compromised system)

Intrusion to install handlers and daemons (continued)

- Subsets with desired architecture selected from compromised systems
- Automated script generated to install handlers programs and DDoS daemons
- Optionally “root kit” installed to hide presence of malicious programs


Client
Client
Handlers
Daemons

Example of DDoS: Trinoo

- First DDoS tool widely known
- Uses UDP flooding as attack strategy
- Access to handlers done via remote TCp connection to the master host
- Master communicates with daemons usnig plain UDP packets

Example of DDoS: Tribe Flood Network

- Written 1999, adds to Trinoo’s UDP flooding
- TCP SYN ICMP flood, smurf attacks
- Handlers accessed using standard TCP connections (telnet, ssh)
- Communication between handler and daemons with ICMP_ECHO_REPLY packets (harder to detect than UDP and can pass firewall systems)

Example of DDoS: TFN2K

- Successor to TFN
- Encrypted communication between components (harder to detect by scanning network)
- Handlers and daemons communicate using either ICMP, UDP, TCP
- Protocol can change for each command, typically selected randomly
- New attack form: TARGA
  - Sends malformed IP packets known to slow down or hangup TCP/IP network stacks
- MIX attack: mixes UDP, SYN, ICMP_ECH)_REPLY flooding

Example of DDoS: stacheldraht

- Based on early TFN versions, effort to eliminate some of TFN’s weak points
- Communication between handlers and daemons is done via ICMP or TCP
- Remote control via simple client with symmetric key encryption with handler
- Similar to TFN: ICMP, UDP, and TCP SYN flood attacks
- Performs daemon updates automatically

What will be next

- DDoS will evolve (have evolved)
- Better encryption
- More attack forms
- Easier, graphical-based tools
- Integration of distribution phase of handler and daemon programs in user interface
- More automation (daemons and handlers get distributed automatically)

Defending against DDoS: Bugs or protocol errors

- Basic security measures
  - Compromised system no longer needs network attacks
- Firewalls to separate interior net, DMZ from Internet
- Intrusion Detection System to notify of unusual activities
- Filtering in firewall (ingress and egress filtering)
- Implement countermeasures for known attacks
- Keep security patches up-to-date

Defending against DDoS: Overload

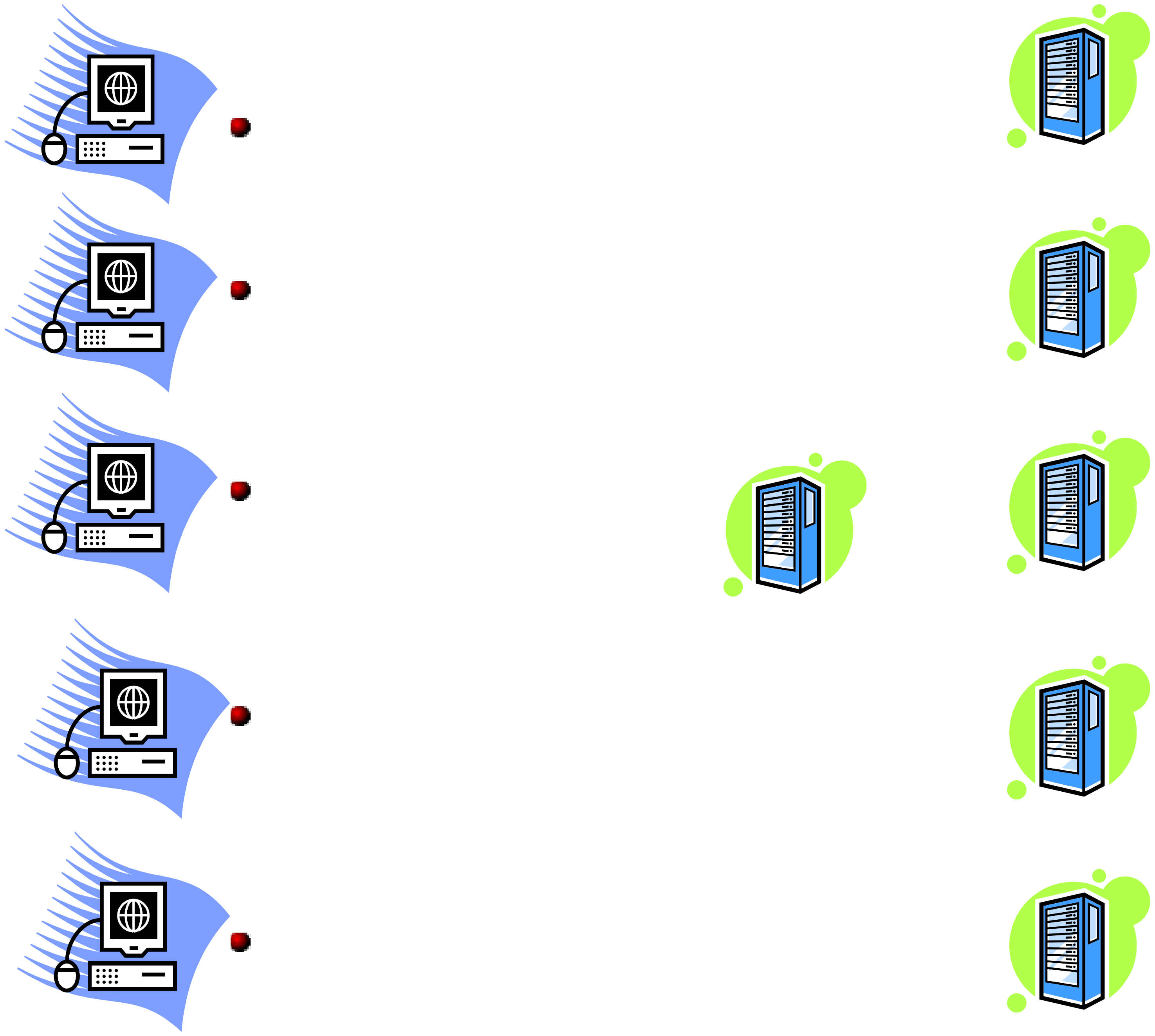
- More difficult
  - Need to distinguish traffic
  - Attack? O r heavy legitimate traffic by many clients?
- One opinion is that global security needs to be applied in Internet
  - Not happening anytime soon
  - Needs co-operation among many entities
- Need defense against DDoS

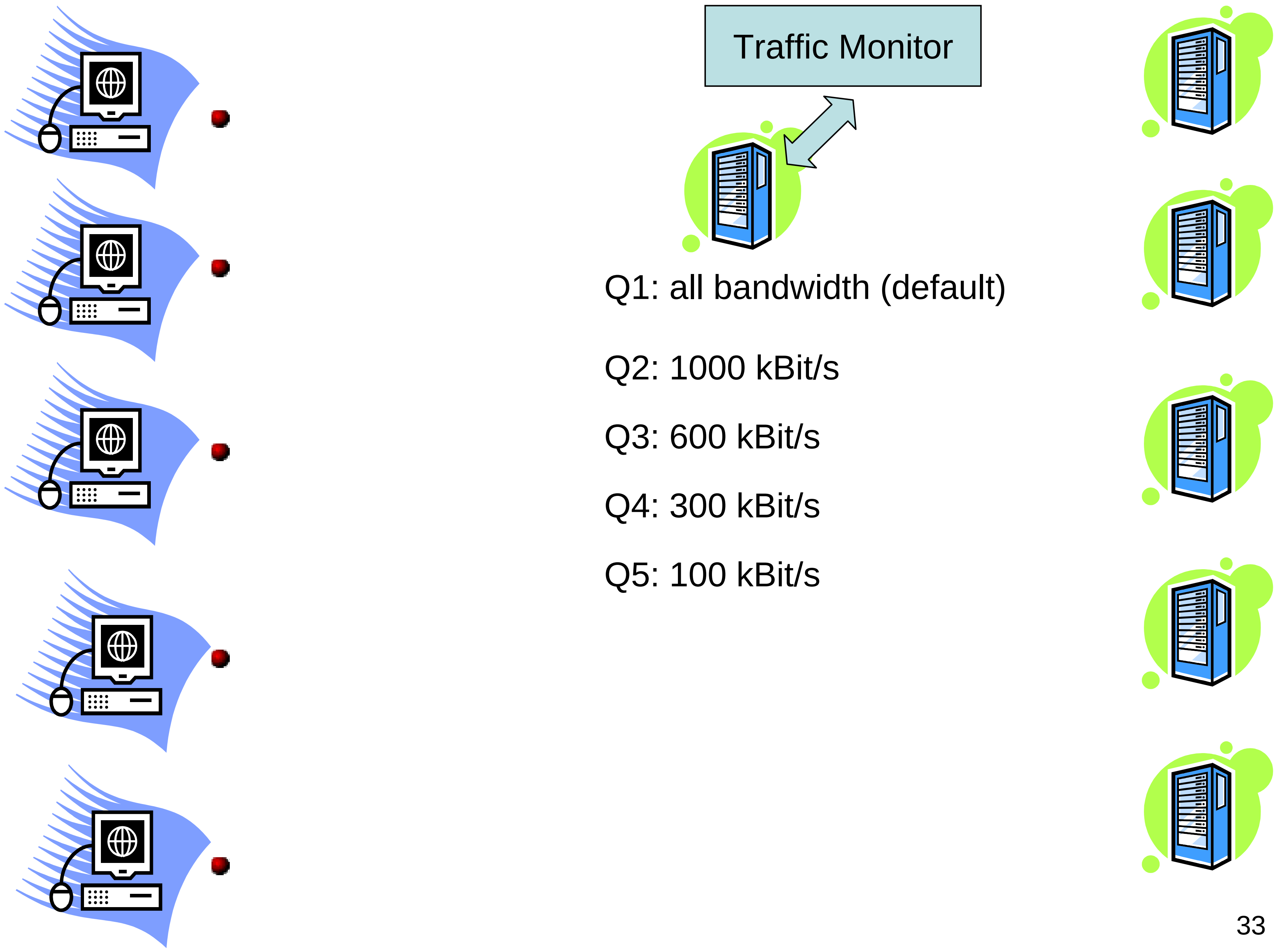
Class Based Queuing (CBQ)

- Function of Linux Kernel
- Setup different traffic queues
- Rules determine which packets go to which queues
- Stochastic Fairness Queuing (SFQO is used for queuing discipline, uses less resources)

Traffic Monitor

- Leverages CBQ
- Monitors for DoS attack
- Consists of a manager and a monitor program
- Monitor runs on the load balancer and all web servers
- Manager runs on server





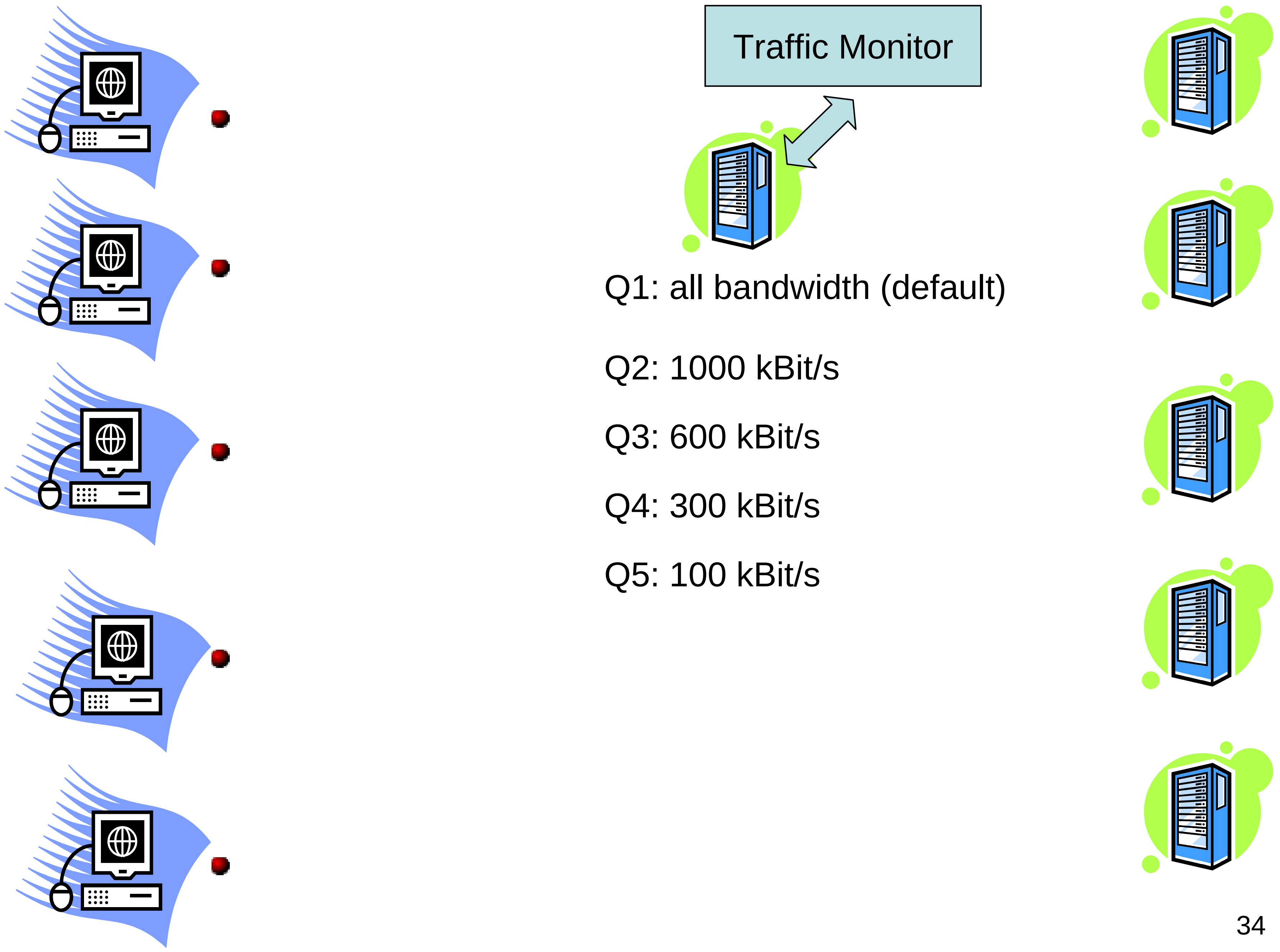
Q1: all bandwidth (default)

Q2: 1000 kBit/s

Q3: 600 kBit/s

Q4: 300 kBit/s

Q5: 100 kBit/s



Q1: all bandwidth (default)

Q2: 1000 kBit/s

Q3: 600 kBit/s

Q4: 300 kBit/s

Q5: 100 kBit/s

Traffic Monitor

- DDoS traffic is slowed down, or stopped
- Key benefit: normal traffic is still allowed to go through at a normal (or near-normal) speed
- “Automatic traffic shaping”

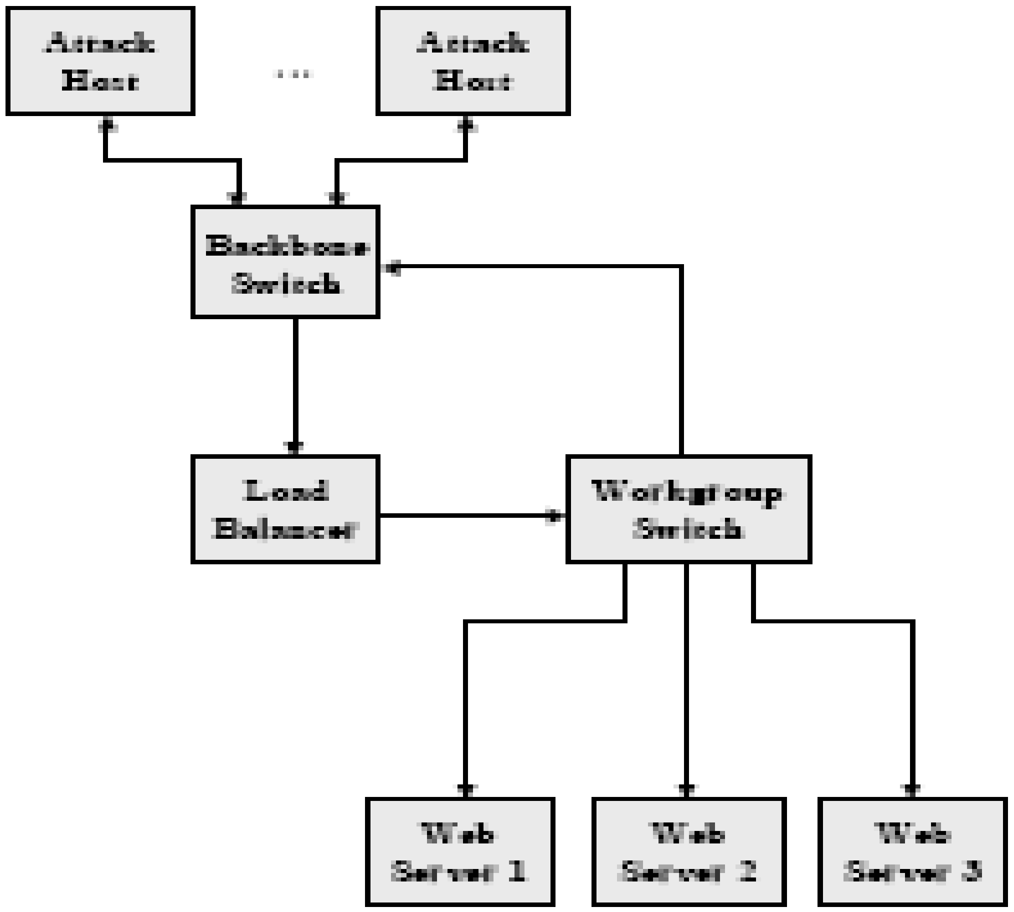
Traffic Monitor: Monitor

- Monitor implemented as 3 separate threads
- Thread 1 monitors the network for packets from and to web server addresses
  - Source IP address, length and time of occurrence are noted in hash table (IP address as key)
- Thread 2 checks hash table every 3 seconds
  - Looks for IP addresses emitting or receiving too much traffic
  - Looks for packet/size ratio’s under a certain amount
  - Flags these IP addresses as potential attacker
- Thread 3 listens to commands from manager
  - Polls all monitors at regular intervals for list of potential attackers
  - Decides if attacker, if so categorizes and downgrades

Traffic Monitor: Attacker classes

- Manager assigns attacking IP’s to a class
- Class 1: IP’s have produced to much traffic with very small packets.
  - Considered very likely a DDoS, block
- Class 2: Too much bandwidth over a long time
  - Downgraded to a lower queue
- Class 3: Too much bandwidth but only for a short time
  - Could be peak from ordinary client
  - Under suspicion
  - Timer started, if behavior continues, put into Class 2
- Class 4: don’t produce or consume much bandwidth but send lots of small packets
  - Again put under suspicion, timer started
- All queues and filters have associated expiration timers
  - After expiration, filter deleted
  - After expiration, IP upgraded to next highest queue class

Performance Tests



Performance Tests (continued)

- Eight attack hosts
- One “normal” client (browsing behavior)
- Goal: show that normal client is unaffected by DDoS
- Attack hosts could produce 8x input capacity of the load balancer

Performance Tests (continued)

- “http_load” (Jef Poskanzer) to simulate attacks
  - Runs multiple HTTP fetches
  - Reads 100 URL’s from file, fetch randomly
  - 64 threads in parallel, try to fetch as many pages as possible in 210 seconds
  - URL’s include static HTML, CGI scripts

Performance Tests (continued)

- Other test tools:
  - TFN2K
  - SYN Flooder
    - Performs a heavy SYN-flood attack
    - Slightly modified to generate random IP addresses
- “http_load” to simulate normal web client
- Sniffers inserted at 3 positions to measure traffic

Test 1: http-attack using http_load and static html database

- Without traffic monitor, response degrades until unusable
- With traffic monitor, system degrades attacking hosts to restricted queues
  - Normal client is able to access web server

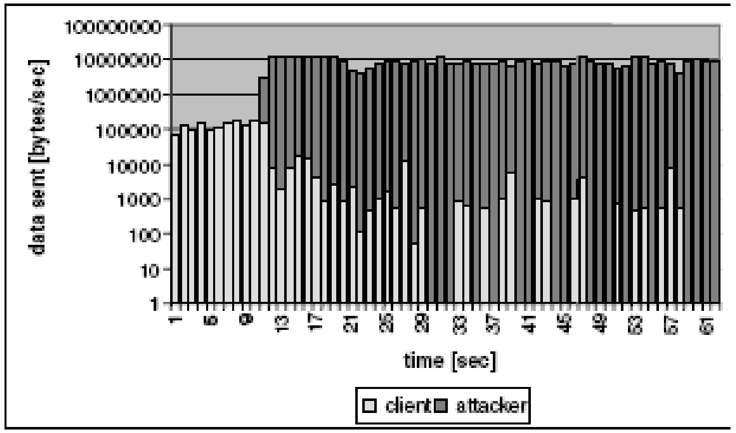


Figure 5: Test 1 without traffic monitor

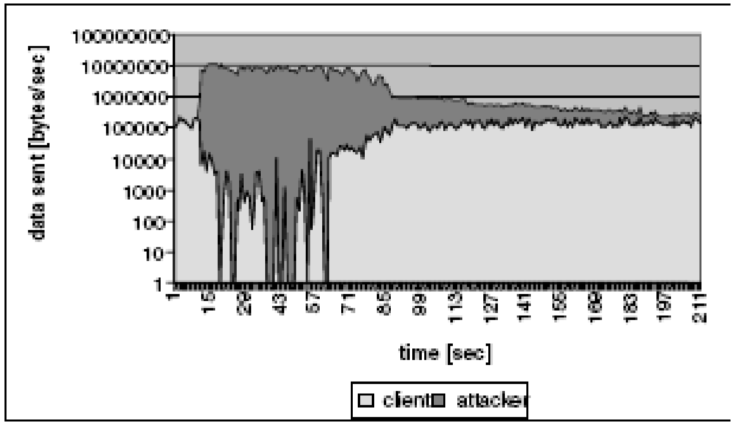


Figure 6: Test 1 with traffic monitor

Test 2: http-attack using http_load and CGI-database

- Instead of static HTML documents, serve documents generated via a CGI script
- Overall traffic much lower than Test 1
  - CGI scripts are CPU intensive
  - Single attacker produces less traffic, less than threshold to classify as attacker
  - Attacker's aren't recognized
  - No sensor for CPU load in system
  - Shows that to react, must be able to distinguish attacks from normal behavior

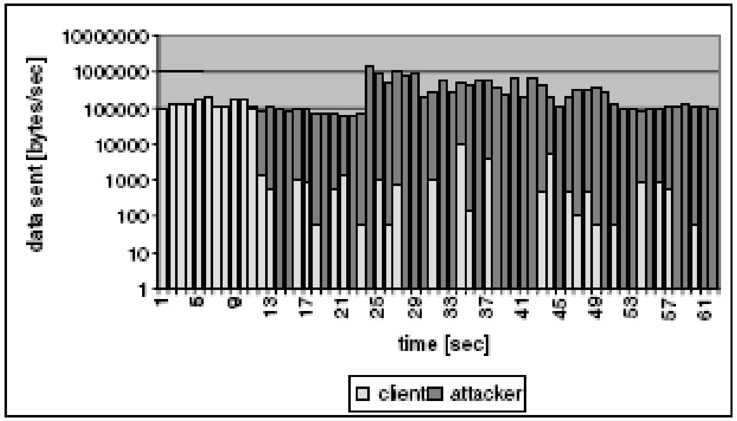


Figure 7: Test 2 without traffic monitor

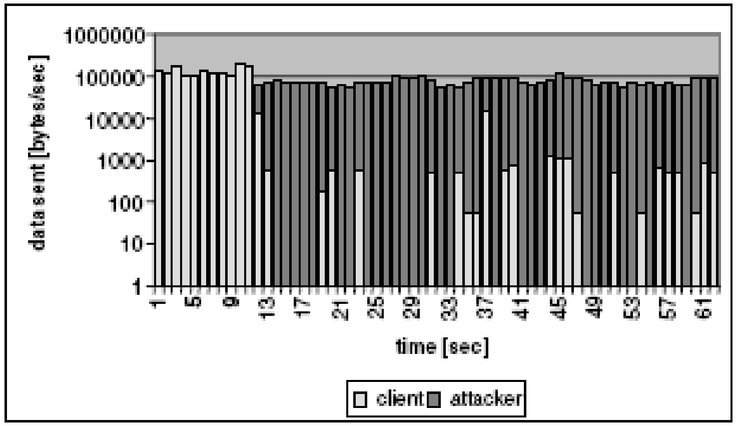


Figure 8: Test 2 with traffic monitor

Conclusions

- Automatic traffic monitoring and shaping is a promising approach: allows normal traffic to go through
- More work needed to detect intrusions?
- DDoS defense scheme must be part of overall security policy
- Paper could have been better done (mislabeled figures, etc.)