TCP and UDP port usage

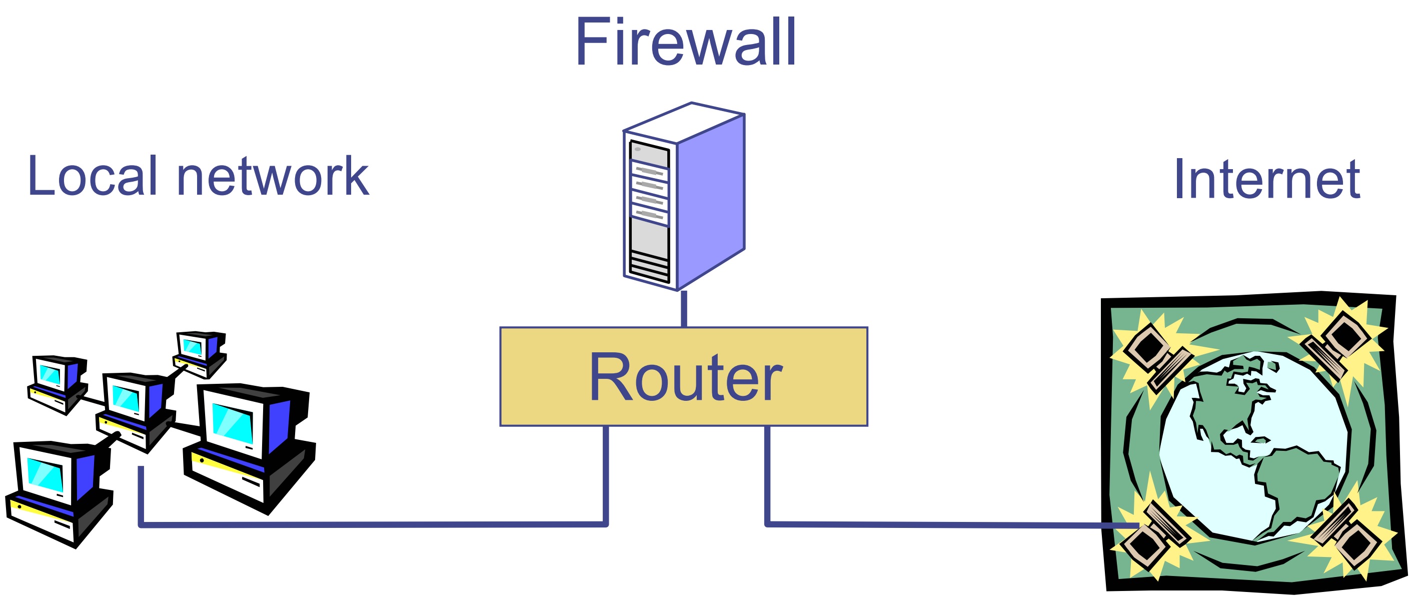
- **Well known services typically run on low ports** < 600
- **Privileged RPC servers us ports** $< 1,024$
  - On Unix must be root to bind port numbers below 1,024
- **Outgoing connections typically use high ports**
  - Usually just ask OS to pick an unused port number
  - Some clients use low ports to "prove" they are root
    E.g., NFS mount client must use reserve port
- **Some applications also use high ports**
  - E.g., X-windows uses port 6,000, NFS port 2,049,
    web proxies on port 3,128
- **See file** `/etc/services` **for well know ports**

Insecure network services

- **NFS (port 2049)**
  - Read/write entire FS as any non-root user given a dir. handle
  - Many OSes make handles easy to guess
- **Portmap (port 111)**
  - Relays RPC requests, making them seem to come from localhost
  - E.g., old versions would relay NFS mount requests
- **FTP (port 21) – server connects back to client**
  - Client can specify third machine for "bounce attack"
- **YP/NIS – serves password file, other info**
- **A host of services have histories of vulnerabilities**
  - DNS (53), rlogin (513), rsh (514), NTP (123), lpd (515), …
  - Many on by default—compromised before OS fully installed

Firewalls

- **Separate local area net from Internet**
  - Prevent bad guys from interacting w. insecure services
  - Perimeter-based security

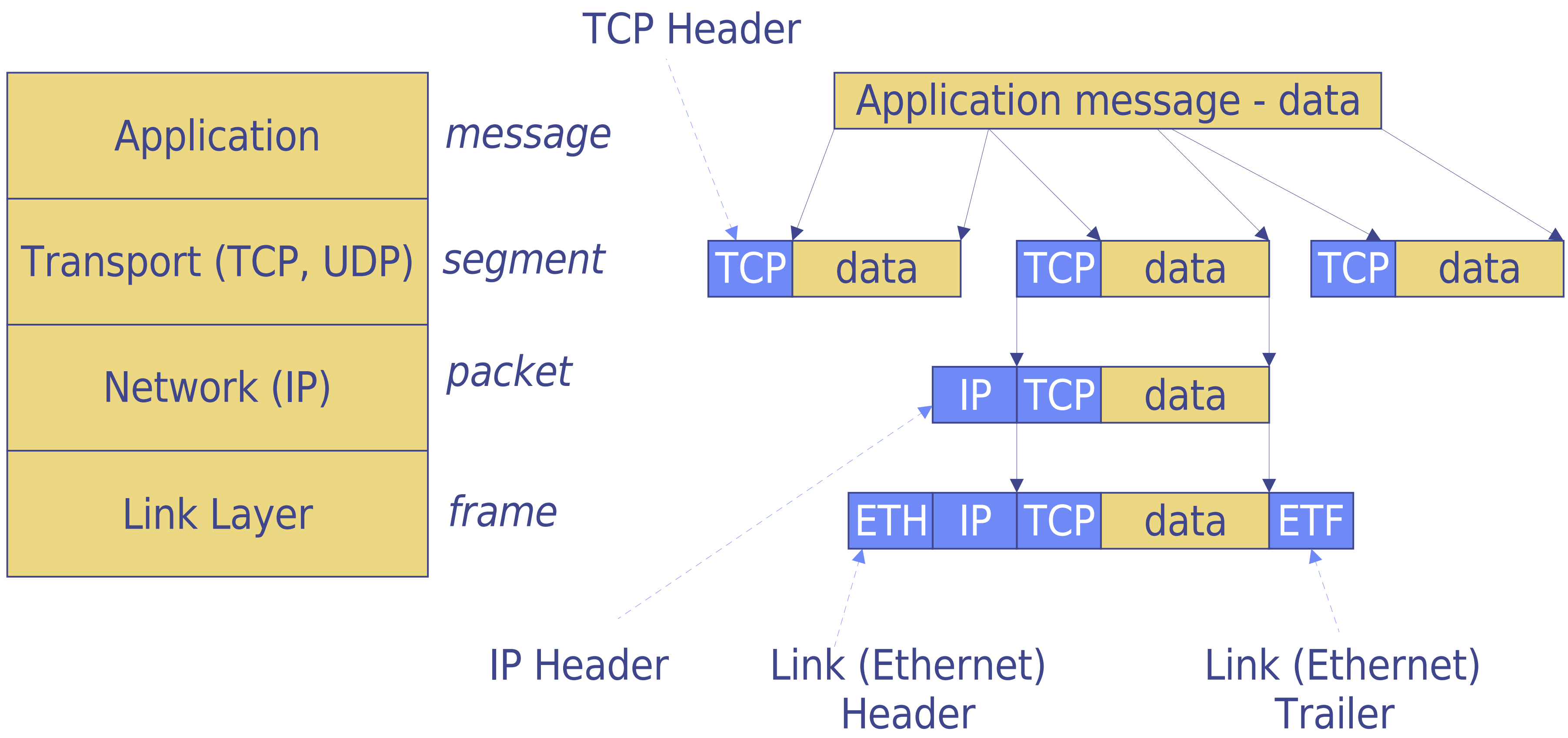


All packets between LAN and internet routed through firewall

Two separable topics

- **Arrangement of firewall and routers**
  - Separate internal LAN from external Internet
  - Wall off subnetwork within an organization
  - Intermediate zone for web server, etc.
  - Personal firewall on end-user machine
- **How the firewall processes data**
  - Packet filtering router
  - Application-level gateway
    Proxy for protocols such as ftp, smtp, http, etc.
  - Personal firewall
    E.g., disallow telnet connection from email client

Recall protocol layering

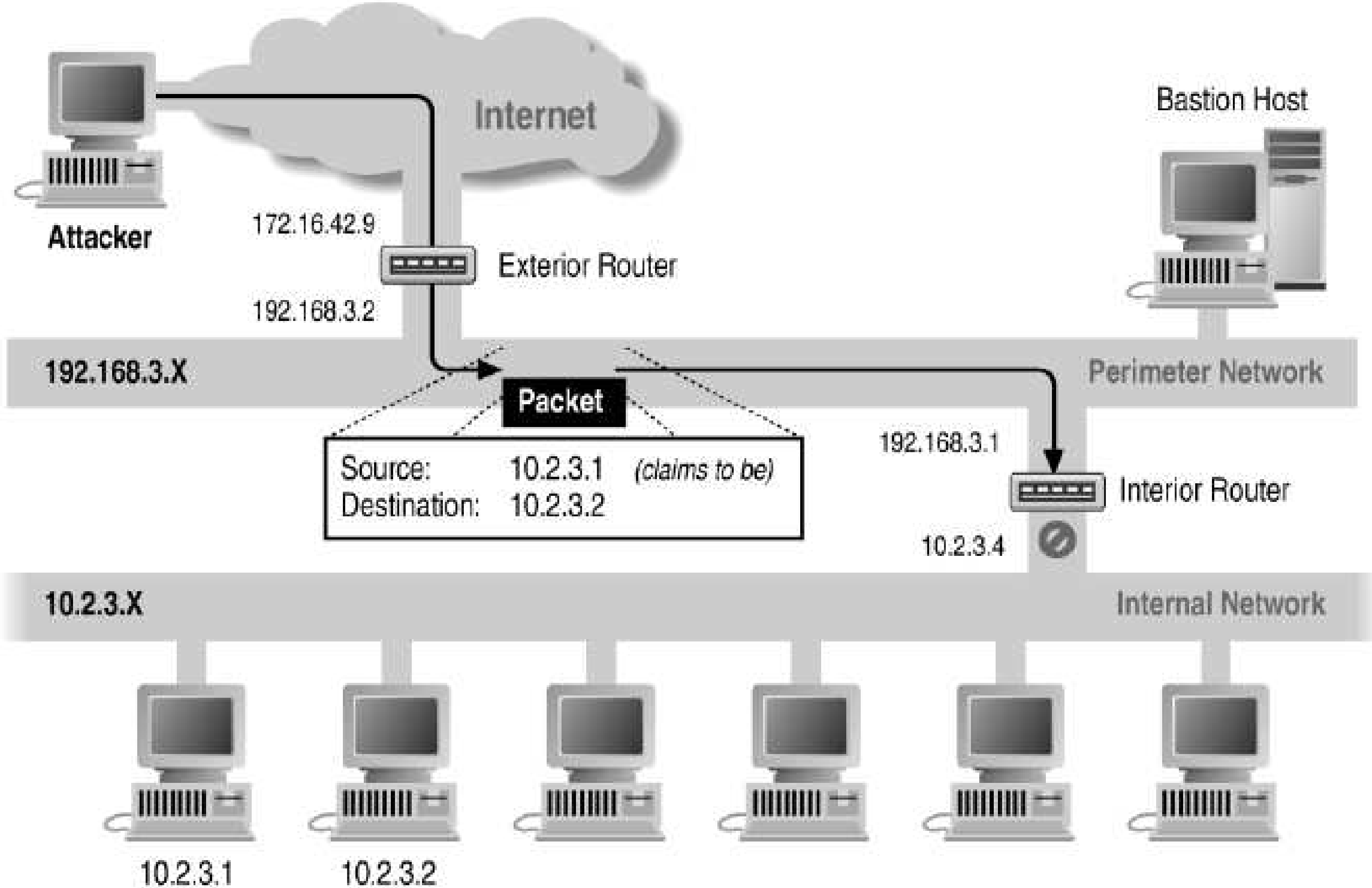


- **E.g., HTTP on TCP on IP on Ethernet**

Packet filtering

- **Filter packets using transport layer information**
  - Examine IP, and ICMP/UDP/UDP header of each packet
  - IP Source, Destination address
  - Protocol
  - TCP/UDP source & destination ports
  - TCP flags
  - ICMP message type
- **Example: coping with vulnerability in lpd**
  - Block any TCP packets with destination port 515
  - Outsiders shouldn't be printing within net anyway

Example: blocking forgeries



- **Should block incoming packets "from" your net**
- **Egress filtering: block forged outgoing packets**

Example: blocking outgoing mail

- **At Stanford, all mail goes out through main servers**
  - Result of worm that mailed users' files around as attachments
  - Could have disclosed sensitive information
  - Also reduces thread of Stanford being used to spam
- **How to enforce?**

Example: blocking outgoing mail

- **At Stanford, all mail goes out through main servers**
  - Result of worm that mailed users’ files around as attachments
  - Could have disclosed sensitive information
  - Also reduces thread of Stanford being used to spam
- **How to enforce?**
- **Block outgoing TCP packets**
  - If destination port is 25 (SMTP – mail protocol)
  - And if source IP address is not a Stanford server

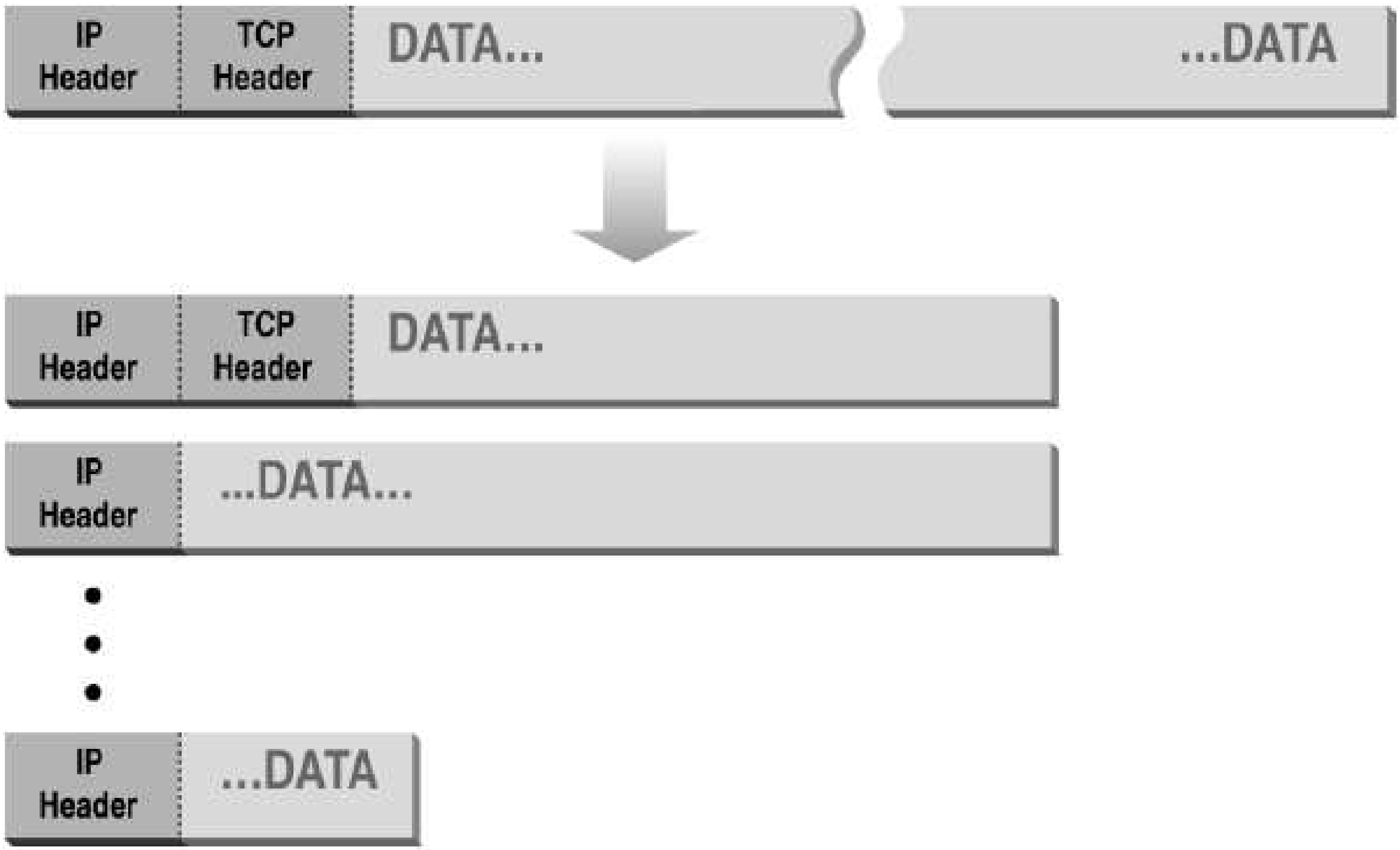
Blocking by default

- **Often don't know what people run on their machines**
- **In many environments better to be safe:**
  - Block all incoming TCP connections
  - Explicitly allow incoming connections to particular hosts
    E.g., port 80 on web server, port 25 on mail server, ...
  - But still must allow *outgoing* TCP connections
    (users will revolt if they can't surf the web)
- **How to enforce?**

Blocking by default

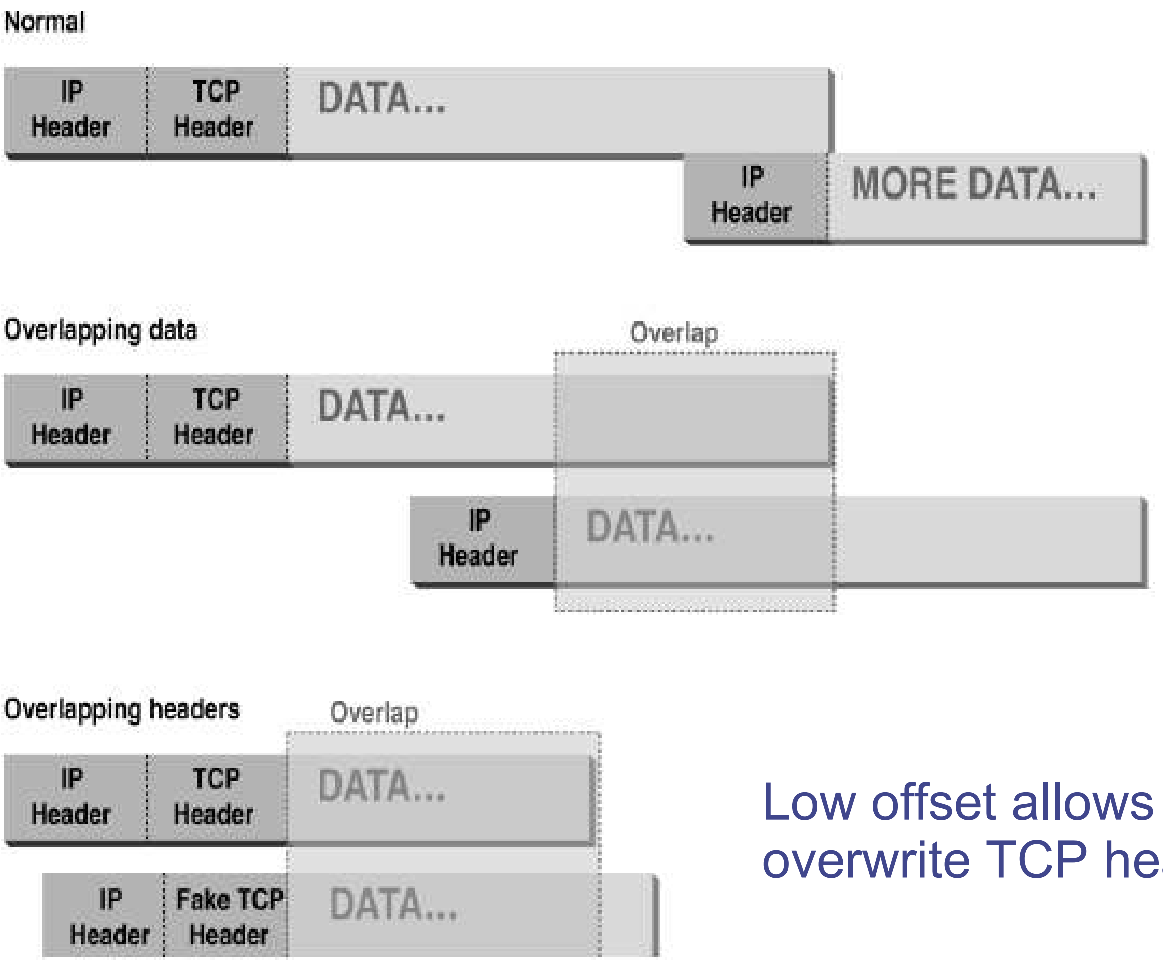
- **Often don't know what people run on their machines**
- **In many environments better to be safe:**
  - Block all incoming TCP connections
  - Explicitly allow incoming connections to particular hosts
    E.g., port 80 on web server, port 25 on mail server, ...
  - But still must allow *outgoing* TCP connections
    (users will revolt if they can't surf the web)
- **How to enforce?**
  - Recall every packet in TCP flow except first has ACK
  - Block incoming TCP packets w. SYN flag but not ACK flag

Fragmentation



- **Recall IP fragmentation—Why might this complicate firewalls?**

Abnormal fragmentation



Low offset allows second packet to overwrite TCP header at receiving host

Fragmentation attack

- **Firewall config: block TCP port 23, allow 25**
- **First packet**
  - Fragmentation Offset = 0.
  - DF bit = 0 : "May Fragment"
  - MF bit = 1 : "More Fragments"
  - Dest Port = 25 (allowed, so firewall forwards packet)
- **Second packet**
  - Frag. Offset = 1: (overwrites all but first byte of last pkt)
  - DF bit = 0 : "May Fragment"
  - MF bit = 0 : "Last Fragment."
  - Destination Port = 23 (should be blocked, but sneaks by!)
- **At host, packet reassembled and received at port 23**

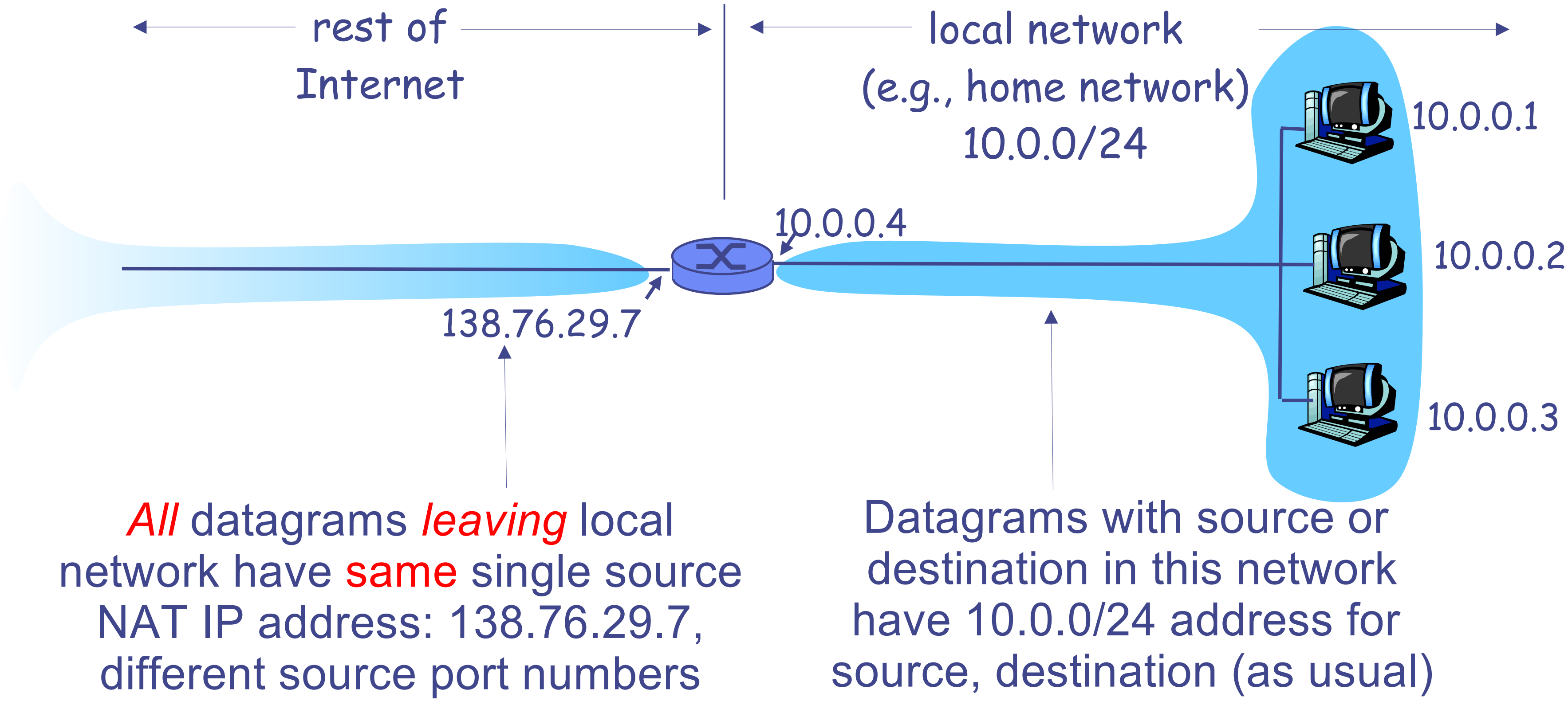
Blocking UDP traffic

- **Some sites block most UDP traffic**
  - UDP sometimes viewed as “more dangerous”
  - Easier to spoof source address
  - Used by insecure LAN protocols such as NFS
- **Often more convenient to block only *incoming* UDP**
  - E.g., allow internal machines to query external NTP servers
  - Don’t let external actors to exploit bugs in local NTP software (unless client specifically contacts bad/spoofed server)
- **How to implement?**

Blocking UDP traffic

- **Some sites block most UDP traffic**
  - UDP sometimes viewed as “more dangerous”
  - Easier to spoof source address
  - Used by insecure LAN protocols such as NFS
- **Often more convenient to block only *incoming* UDP**
  - E.g., allow internal machines to query external NTP servers
  - Don’t let external actors to exploit bugs in local NTP software (unless client specifically contacts bad/spoofed server)
- **Must keep state in firewall**
  - Remember ⟨local IP, local port, remote IP, remote port⟩ for each outgoing UDP packet
  - Allow incoming packets that match saved flow
  - Time out flows that have not been recently used

Network address translation (NAT)



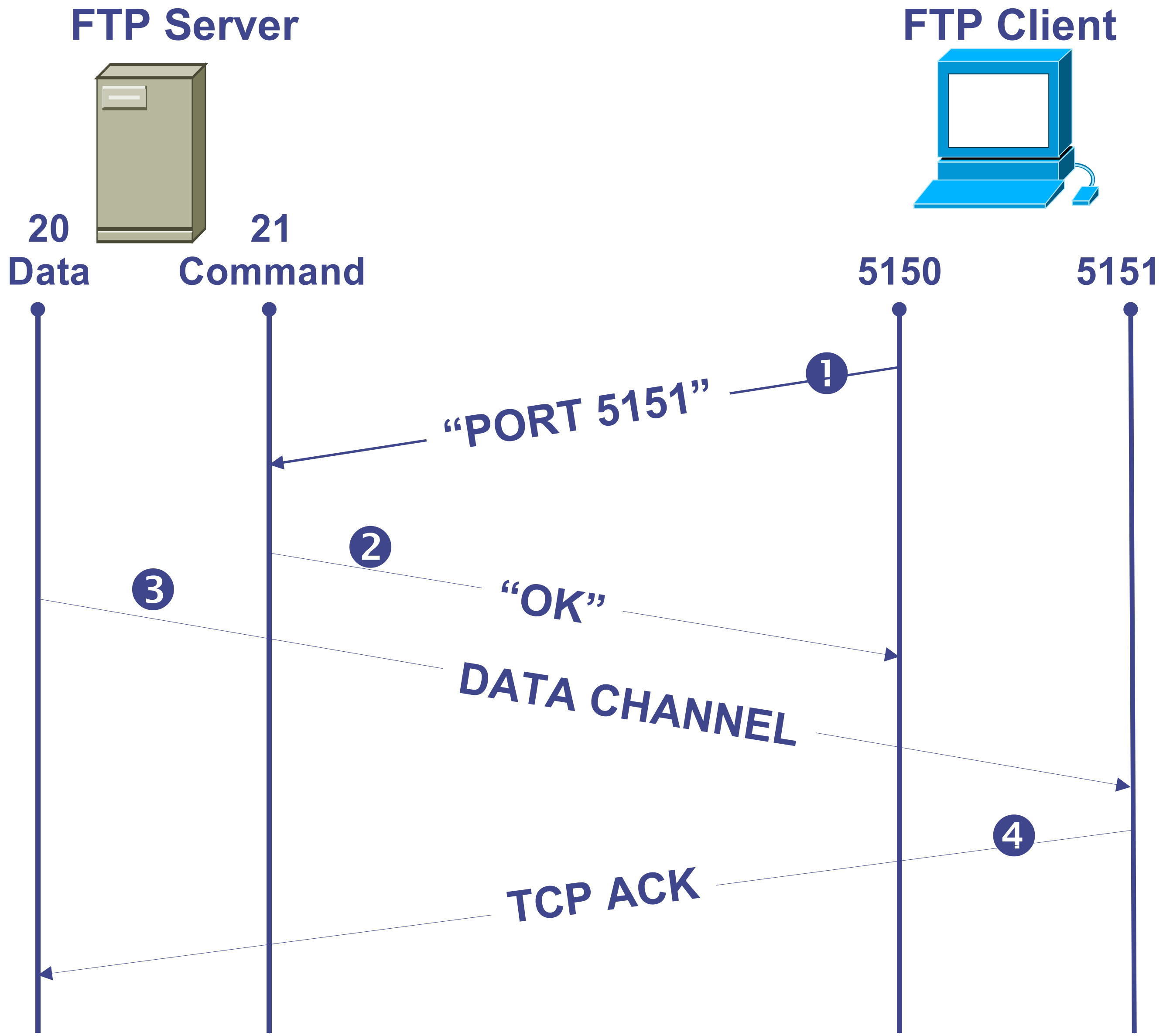
[Kurose and Ross]

- **NAT translates from private IP addresses to public**
- **Similarly must keep state for each flow**

Advantages of NAT

- **Motivations for NAT**
  - Have more machines than public IP addresses
  - Easy way to get "no incoming flows" policy
  - Avoid renumbering if provider changes (Small/mid-sized LANs inherit address space from ISP)
- **Hides information about internal net from server**
- **Can "scrub" packets to further enhance security**
  - Exact SYN packet format may reveal OS & version
  - Map predictable TCP Seq No's to unpredictable ones
  - OpenBSD's pf "modulate state" option good at this

How to firewall FTP protocol?



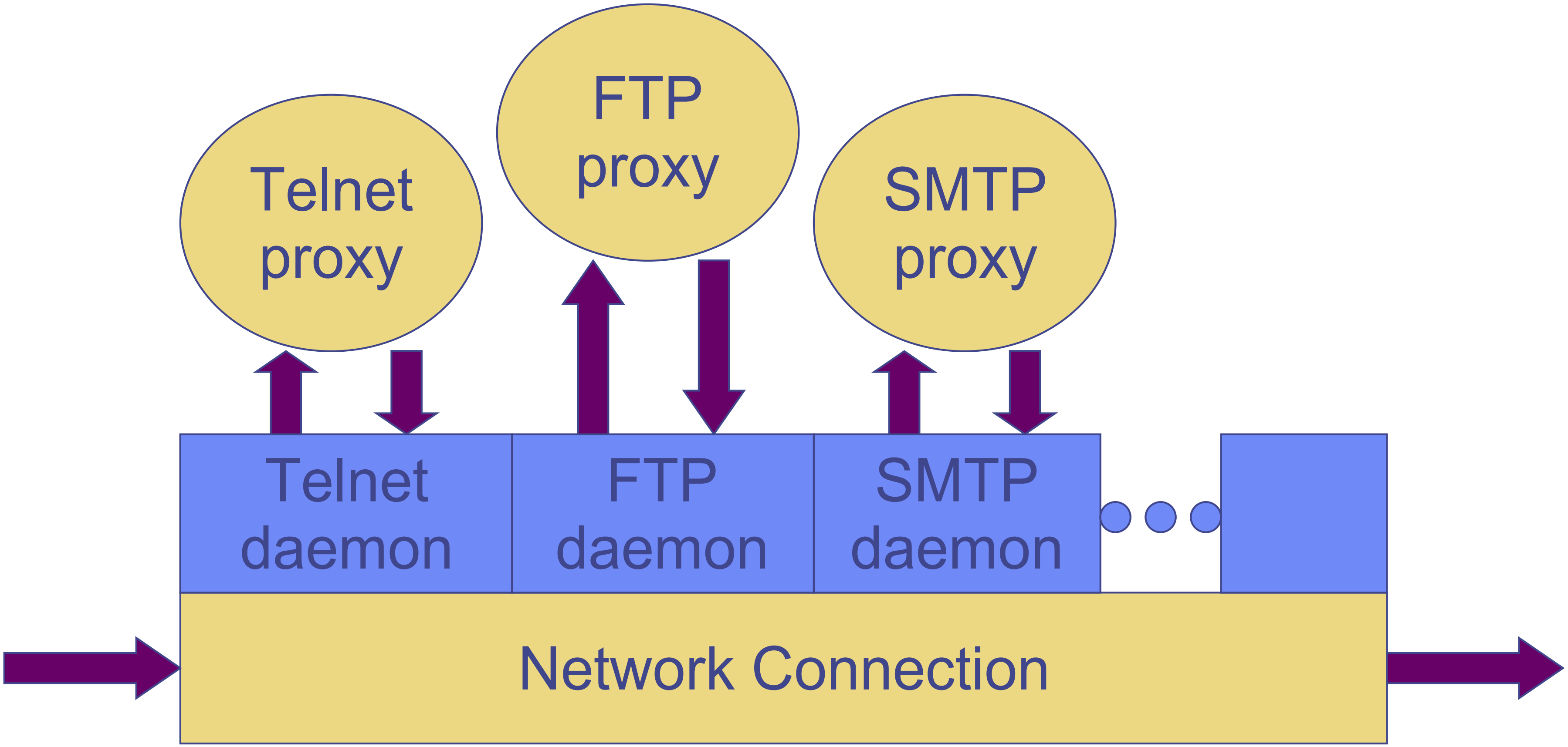
❶ Client opens command channel to server; tells server second port number

❷ Server acknowledges

❸ Server opens data channel to client's second port

❹ Client acknowledges

Application proxies on firewall

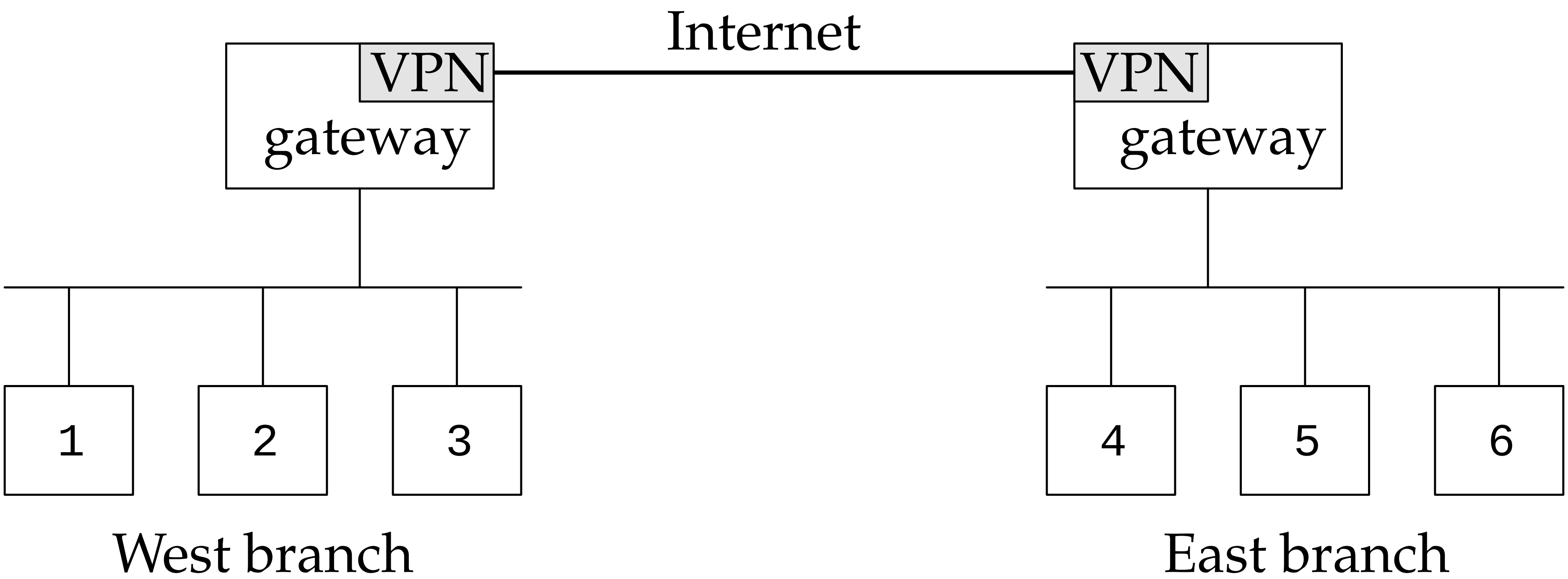


- **Spawn proxy on firewall when connection detected**

Application-level proxies

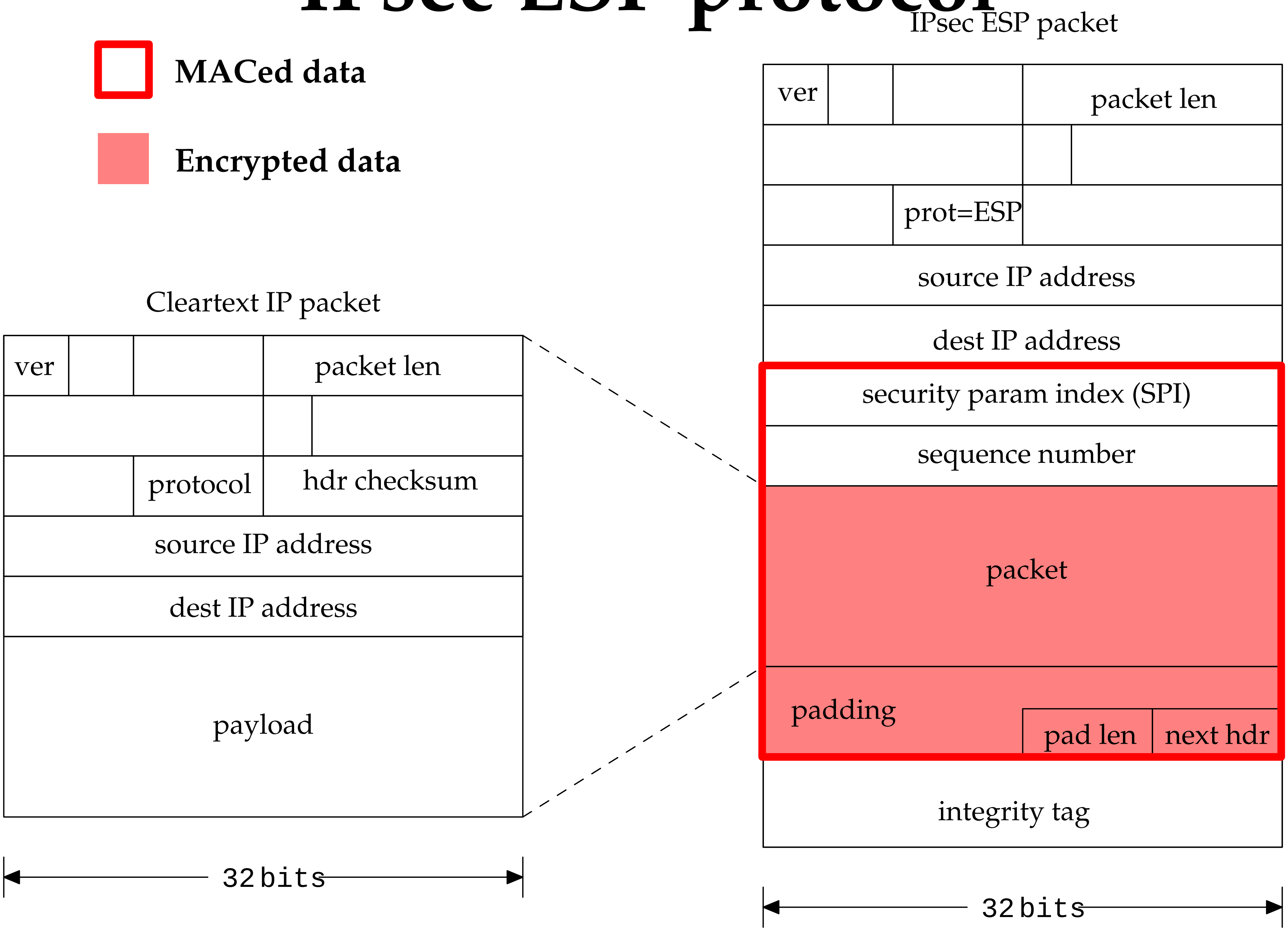
- **Enforce policy for specific protocols**
  - E.g., Virus scanning for SMTP, must understand MIME, encoding, Zip archives, etc.
  - Flexible approach, but may introduce network delays
- **Many protocols natural to proxy**
  - SMTP, NNTP (Net news), DNS, NTP, HTTP
- **But sometimes results in weird artifacts**
  - E.g., caching HTTP objects unexpectedly
- **Encrypted protocols typically not: SSL, SSH**
- **Must protect host running protocol stack**
  - Much more complexity than simple packet filter
  - Be prepared for the system to be compromised

Virtual Private Networks (VPNs)



- **What if firewall must protect more than one office**
- **Extend perimeter to other physical networks by using crypto – VPN**
- **Two popular VPNs: IPsec & OpenVPN [SSL-based]**

IPsec ESP protocol



ESP high-level view

- **Encapsulates one IP packet inside another**
- **Each endpoint has *Security Association DB* (SAD)**
  - Is a table of *Security Associations* (SAs)
  - Each SA has 32-bit *Security Parameters Index* (SPI)
  - Also, source/destination IP addresses, crypto algorithm, keys
- **Packets processed based on SPI, src/dest IP address**
  - Usually have one SA for each direction betw. two points
- **SAD managed “semi-manually”**
  - Manually set key
  - Or negotiate it using IKE protocol

ESP details

- **Must avoid replays**
  - Keep counter for 64-bit sequence number
  - Receiver must some packets out of order (e.g., up to 32)
  - Only low 32 bits of sequence number in actual packet (would be bad if you lost 4 billion packets)
- **Support for traffic flow confidentiality (TFC)**
  - Can pad packets to fixed length
  - Can send dummy packets
- **Support for encryption without MAC…Bummer!**
  - Rationale: App might be SSL, which has MAC-only mode
  - But then attacker can mess with destination address!

Traffic shaping

- **Traditional firewall: Allow or drop each packet**
- **Traffic shaping:**
  - Limit certain kinds of traffic
  - Can differentiate by host addr, protocol, etc
  - Multi-Protocol Label Switching (MPLS): Label traffic flows at the edge of the network and let core routers identify the required class of service
- **The real issue here on Campus:**
  - P2P file sharing takes a lot of bandwidth
  - 1/3 of network bandwidth consumed by BitTorrent (Hmm... What do you guys use BitTorrent for?)

Bro: Detecting network intruders

- **Target security holes exploited over the network**
  - Buffer overruns in servers
  - Servers with bad implementations
    ("login -froot", telnet w. LD_LIBRARY_PATH)
- **Goal: Detect people exploiting such bugs**
- **Detect activities performed by people who've penetrated server**
  - Setting up IRC bot
  - Running particular commands, etc.

Bro model

- **Attach machine running Bro to "DMZ"**
  - Demilitarized zone – area betw. firewall & outside world
- **Sniff all packets in and out of the network**
- **Process packets to identify possible intruders**
  - Secret, per-network rules identify possible attacks
  - Is it a good idea to keep rules secret?
- **React to any threats**
  - Alert administrators of problems in real time
  - Switch on logging to enable later analysis of potential attack
  - Take action against attackers – E.g., filter all packets from host that seems to be attacking

Goals of system

- **Keep up with high-speed network**
  - No packet drops
- **Real-time notification**
- **Separate mechanism from policy**
  - Avoid easy mistakes in policy specification
  - So different sites can specify "secret" policies easily
- **Extensibility**
- **Resilience to attack**

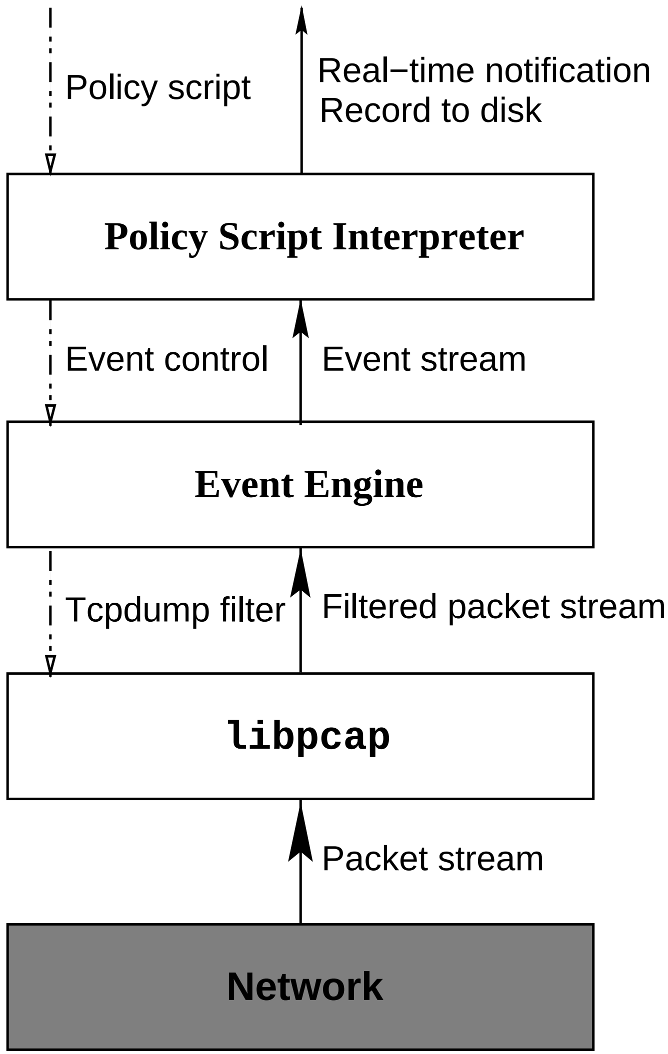
Challenges

- **Have to keep up with fast packet rate**
- **System has to be easy to program**
  - Every site needs different, secret rules
  - Don’t want system administrators making mistakes
- **Overload attacks**
- **Crash attacks**
- **Subterfuge attacks**

Bro architecture

- **Layered architecture:**
  - bpf/libpcap, Event Engine, Policy Script Interpreter
- **Lowest level bpf filter in kernel**
  - Match interesting ports or SYN/FIN/RST packets
  - Match IP fragments
  - Other packets do not get forwarded to higher levels
- **Event engine, written in C++**
  - Knows how to parse particular network protocols
  - Has per-protocol notion of events
- **Policy Script Interpreter**
  - Bro language designed to avoid easy errors

Bro picture


Policy script
Real−time notification
Record to disk
Policy Script Interpreter
Event control
Event stream
Event Engine
Tcpdump filter
Filtered packet stream
libpcap
Packet stream
Network

Overload and Crash attacks

- **Overload goal: prevent monitor from keeping up w. data stream**
  - Leave exact thresholds secret
  - Shed load (e.g., HTTP packets)
- **Crash goal: put monitor out of commission**
  - E.g., run it out of space (too much state)
  - Watchdog timer kills & restarts stuck monitor
  - Also starts tcpdump log, so same crash attack, if repeated, can be analyzed

Challenges

- **Dealing with FTP**
  - Separate pipelined requests
  - Parse PORT command to detect "bounce" attacks
- **Dealing with type-ahead and rejected logins with telnet/rlogin**
  - Flows basically unstructured–don't know what's username
  - Use heuristics (e.g., look for "Password:" string)
  - But typeahead makes it harder to match exactly
- **Network scans and port scans…How to detect**
  - Keep table of connection attempts (src, dst, bool)
  - If not seen yet, increment count of distinct_peers[src]
  - Trade-off between state recovery & detection of slow scans

Subterfuge attacks

- **IP fragments too small to see TCP header**
- **Retransmitted IP fragments w. different data**
- **Retransmitted TCP packets w. different data**
- **Checksum/TTL/MTU monkeying can hide packets from destination**
  - Compare TCP packet to retransmitted copy
  - Assume one of two endpoints is honest (exploit ACKs)
  - Bifurcating analysis

State and checkpointing

- **Need to keep a lot of session state**
  - Open TCP connections, UDP request-response, IP fragments
  - No timers to garbage collect state
- **Checkpointing the system**
  - Start new copy of monitoring process
  - Kill old copy when new copy has come up to speed
  - Is this ideal?

The Kerberos authentication system

- **Goal: Authentication in "open environment"**
  - Not all hardware under centralized control (e.g., users have "root" on their workstations)
  - Users require services from many different computers (mail, printing, file service, etc.)
- **Model: Central authority manages all resources**
  - Effectivaly manages human-readable names
  - User names: dm, dabo, …
  - Machine names: market, cipher, crypto, …
  - Must be assigned a name to use the system

Kerberos principals

- *Principal*: **Any entity that can make a statement**
  - Users and servers sending messages on network
  - “Services” that might run on multiple servers
- **Every kerberos principal has a key (password)**
- **Central key distribution server (KDC) knows all keys**
  - Coordinates authentication between other principals

Kerberos protocol

- **Goal: Mutually authenticated communication**
  - Two principals wish to communicate
  - Principals know each other by KDC-assigned name
  - Kerberos establishes shared secret between the two
  - Can use shared secret to encrypt or MAC communication (but most services don't encrypt, none MAC)
- **Approach: Leverage keys shared with KDC**
  - KDC has keys to communicate with any principal
- **Let's abstract away broken crypto**
  - Assume each key K has two parts, K_e and K_m.
  - Read $\{\text{msg}\}_K$ as $\langle \text{ENC}(K_e, \text{msg}), \text{MAC}(K_m, \text{ENC}(K_e, \text{msg}))\rangle$

Protocol detail

- **To talk to server s, client c needs key & ticket:**
  - Session key: $K_{s,c}$ (randomly generated key KDC)
  - Ticket: $T = \{s, c, \text{addr}, \text{expire}, K_{s,c}\}_{K_s}$
    (K_S is key s shares with KDC)
  - Only server can decrypt T
- **Given ticket, client creates authenticator:**
  - Authenticator: $T, \{c, \text{addr}, \text{time}\}_{K_{s,c}}$
  - Client must know $K_{s,c}$ to create authenticator
  - T convinces server that $K_{s,c}$ was given to c
- **"Kerberized" protocols begin with authenticator**
  - Replaces passwords, etc.

Getting tickets in Kerberos

- **Upon login, user fetches "ticket-granting ticket"**
  - $c \rightarrow t$: c, t (*t* is name of TG service)
  - $t \rightarrow c$: $\{K_{c,t}, T_{c,t} = \{s, t, \text{addr}, \text{expire}, K_{c,t}\}_{K_t}\}_{K_c}$
  - Client decrypts with password ($K_c = H(\text{pwd})$)
- **To fetch ticket for server** s
  - $c \rightarrow t$: $s, T_{c,t}, \{c, \text{addr}, \text{time}\}_{K_{c,t}}$
  - $t \rightarrow c$: $\{T_{s,c}, K_{s,c}\}_{K_{c,t}}$
- **Applications might use Kerberos as follows:**
  - $c \rightarrow s$: $T_{s,c}, \{c, \text{addr}, \text{time}, K_{c\rightarrow s}, K_{s\rightarrow c}\}_{K_{s,c}}$
  - Then c and s use $K_{c\rightarrow s}$ and $K_{s\rightarrow c}$ to communicate securely in each direction.

Example application: AFS

- **User logs in, fetches kerberos ticket for AFS server**
- **Hands ticket and session key to file system**
- **Requests/replies accompanied by an authenticator**
  - Authenticator includes CRC checksum of packets
  - Note: CRC is not a valid MAC!
- **What about anonymous access to AFS servers?**
  - User w/o account may want universe-readable files

AFS permissions

- **Each directory has ACL for all its files**
  - Precludes cross-directory links
- **ACL lists principals and permissions**
  - Both “positive” and “negative” access lists
- **Principals: Just kerberos names**
  - Extra principles, system:anyuser, system:authuser
- **Permissions: rwlidak**
  - read, write, lookup, insert, delete, administer, lock

Security issues with kerberos

Security issues with kerberos

- **Protocol weaknesses:**
  - Weak crypto, no MAC
  - Kinit might act as oracle because of bad MAC
  - Replay attacks
  - Off-line password guessing
  - Can't securely change compromised password
- **General design problems:**
  - KDC vulnerability
  - Hard to upgrade system (everyone relies on KDC)

Kerberos inconvenience

- **Large (e.g., university-wide) administrative realms**
  - University-wide administrators often on the critical path
  - Departments can't add users or set up new servers
  - Can't develop new services without central admins
  - Can't upgrade software/protocols without central admins
  - Central admins have monopoly servers/services
    (Can't set up your own without a principal)
- **Crossing administrative realms a pain**
- **Ticket expirations**
  - Must renew tickets every 12–23 hours
  - Hard to have long-running backround jobs

SSH overview

- **Widely-used secure remote login program**
- **MACs/encrypts all data sent over the network**
  - Version 2 of protocol basically gets this right (should MAC ciphertext not plaintext, but OK w. particular algorithms)
  - Open to man in the middle attack on first server access
- **Often sends password at start of session**
  - Gets sent encrypted in a single TCP packet
- **Assuming crypto secure (& no MiM), how to attack? [Material from Song et. al follows]**

Packet size

- **Transmitted packets rounded to multiple of 8 bytes**
  - Version 1 even had exact packet-size in the clear
- **Can tell if user's password is less than 7 chars**
  - Password sent in one packet of initial exchange
- **Why do we care?**

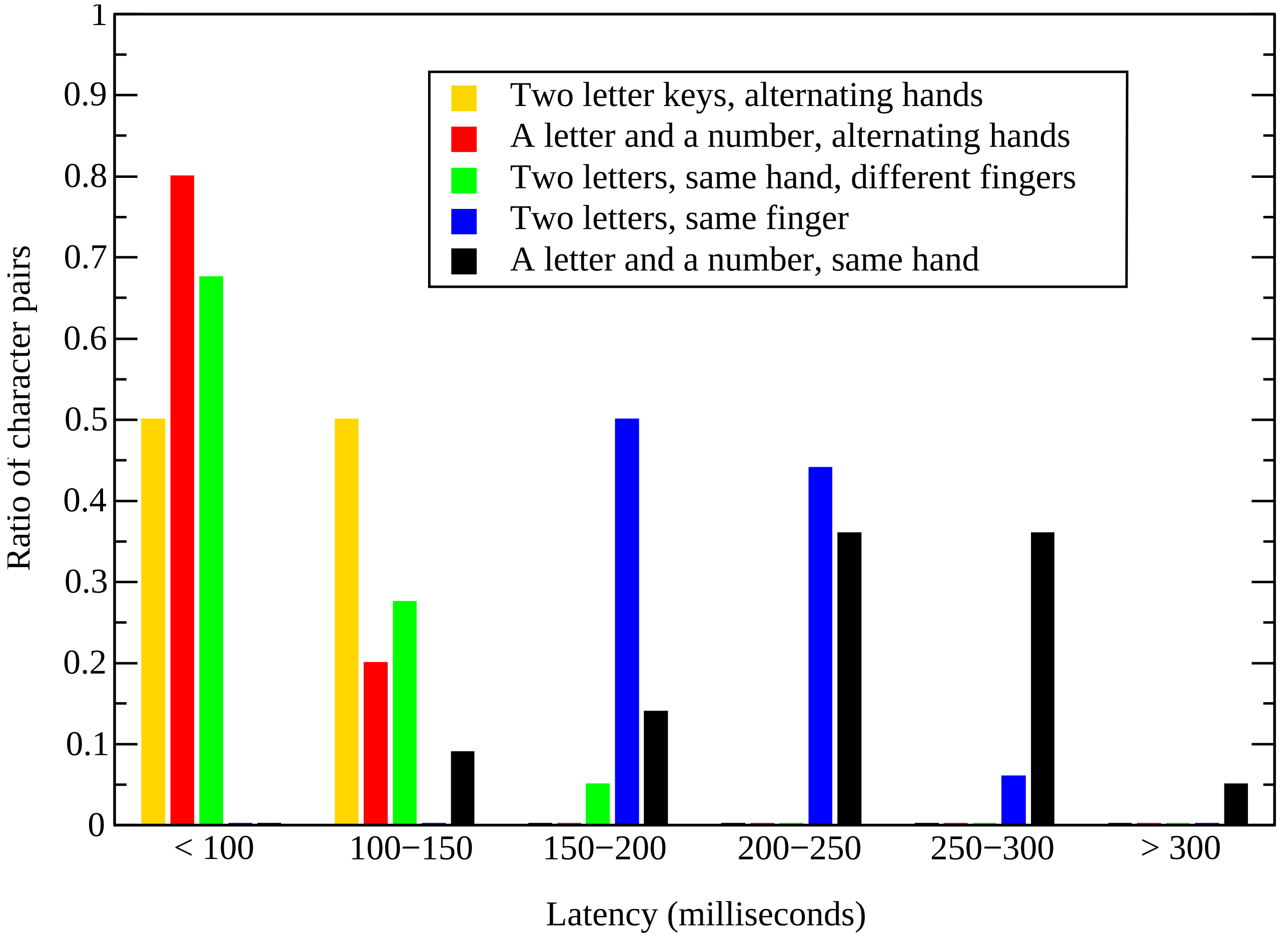
Packet size

- **Transmitted packets rounded to multiple of 8 bytes**
  - Version 1 even had exact packet-size in the clear
- **Can tell if user's password is less than 7 chars**
  - Password sent in one packet of initial exchange
- **Why do we care?**
  - Might tell you which account to try to crack

Inter-keystroke timings

- **Each character typed causes a packet to be sent**
  - Typical inter-character times 10–300 msec
  - Typical network round-trip time 10 of msec
  - Can get very accurate timing information by eavesdropping
- **What can you learn from this?**
  - Some character sequences harder to type than others
  - E.g., v–b is much slower to type than v–o
  - In general, characters with different hands faster
  - Two characters typed with same finger are much slower
  - Digits, special chars also slower
- **Idea: Use timing to learn about passwords**

Character latency

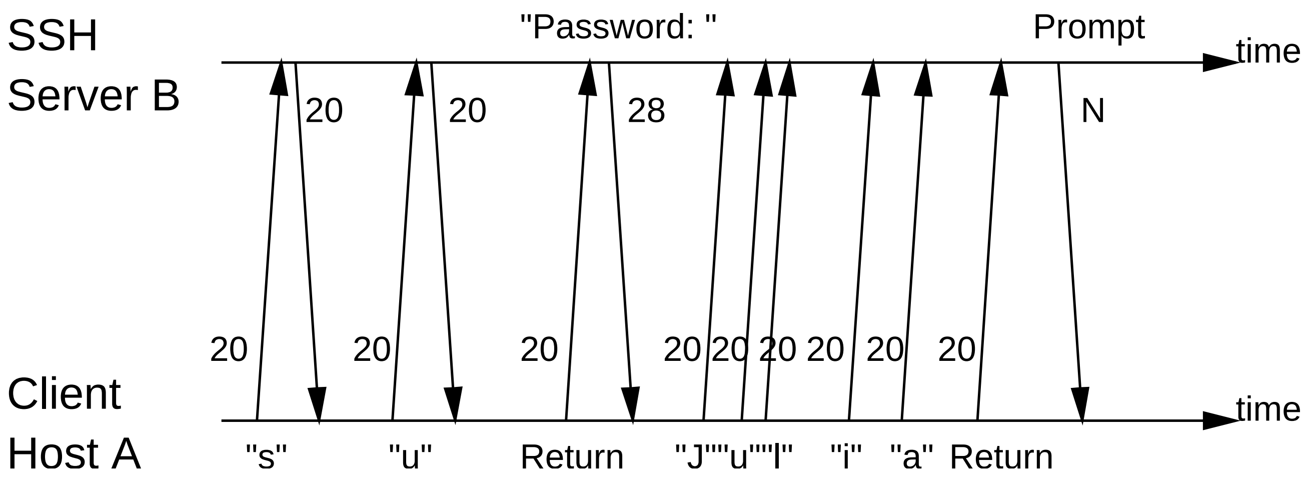


How to know password being typed?

How to know password being typed?

- **Traffic signature**
  - E.g., echo turned off when password typed
- **Multi-user attack**
  - E.g., run ps on machine to see when victim runs pgp
- **Nested ssh attack**
  - See remote host open SSH connection to another host

Example: su command

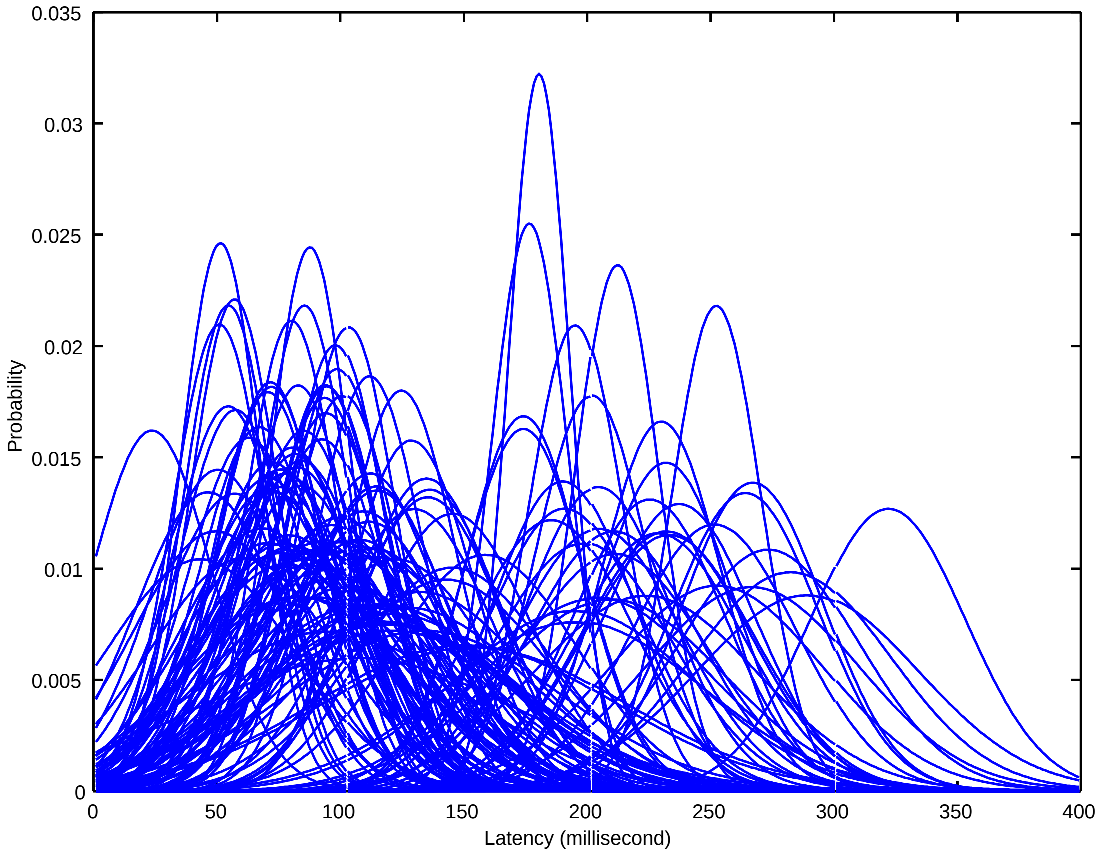


- **"Password:" prompt – 28 char packet**
- **Echo turned off for password, no return packets**

Modeling keystroke timings

- **Assume Gaussian-like distribution of timings**
  - For each key pair q, mean time μ_q, stdev σ_q
  - Prob. of timing y $\Pr[y|q] = \frac{1}{\sqrt{2\pi}\sigma_q} e^{-\frac{(y-\mu_q)^2}{2\sigma_q^2}}$
  - Significant but far from complete overlap between key pairs
- **Model keystrokes as HMM**
  - Each key pair is a state, timing an observation
  - AI techniques allow you to get n best choices

Latency vs. probability of key pairs



Results

- **Experiment: Assign users random passwords**
  - Picked from a reduced set of characters
  - Users practice typing the password before experiments
- **Train on users typing individual key pairs**
- **Ignore pause in the middle of passwords**
- **Output most likely password**
- **Bottom line:** $50\times$ **reduction in brute-force cracking**
  - Half the time password shows up in top 1% output

How to work around the problem

- **Send dummy packets when in echo mode**
  - Foils traffic signature detection of passwords
- **Adding random delays to packets?**
  - Latencies in 100s of msec, so need big random delays
  - Can still get info by averaging many sessions
  - Delay might get seriously annoying
- **Constant bit-rate traffic**
  - Practical for *one session* over a modem