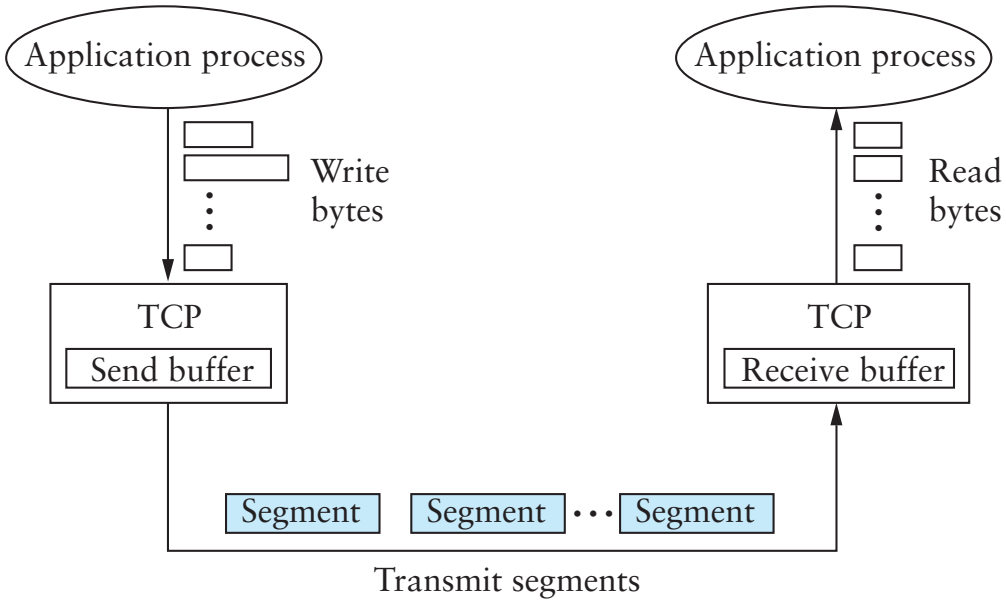
Announcements

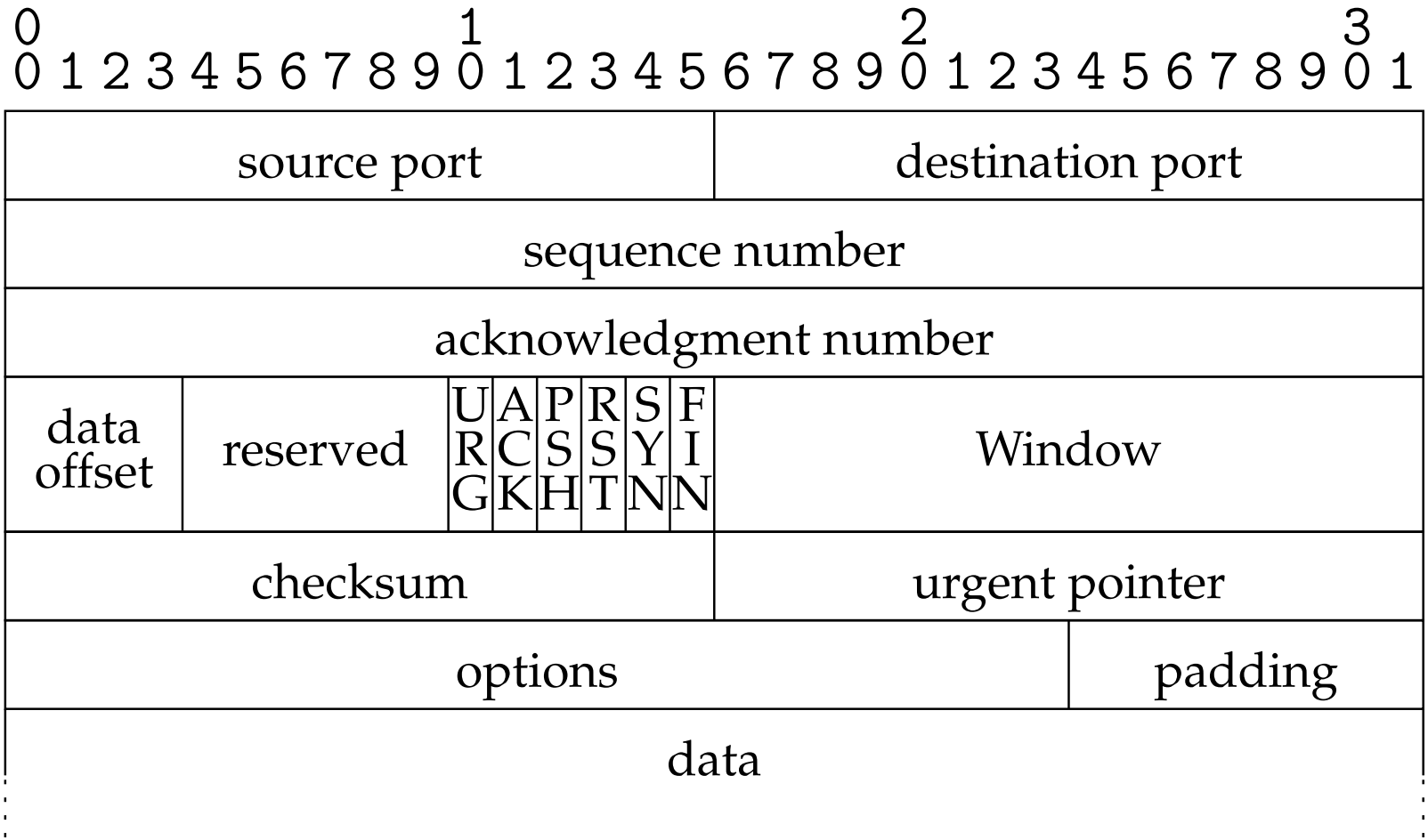
- **My office hours moved to Friday 4pm this week**
- **Lakshminarayan Subramanian talking at NYU**
  - Monday, April 11, 2005 11:15 AM (refreshments 11AM)
  - WWH 1302

Overview of TCP



- **Full duplex, connection-oriented byte stream**
- **Flow control**
  - If one end stops reading, writes at other eventuall block/fail
- **Congestion control**
  - Keeps sender from overrunning network

TCP segment


0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
source port
destination port
sequence number
acknowledgment number
data offset
reserved
URG
ACK
PSH
RST
SYN
FIN
Window
checksum
urgent pointer
options
padding
data

TCP fields

- Ports
- Seq no. – segment position in byte stream
- Ack no. – seq no. sender expects to receive next
- Data offset – # of 4-byte header & option words
- Window – willing to receive (flow control)
- Checksum
- Urgent pointer

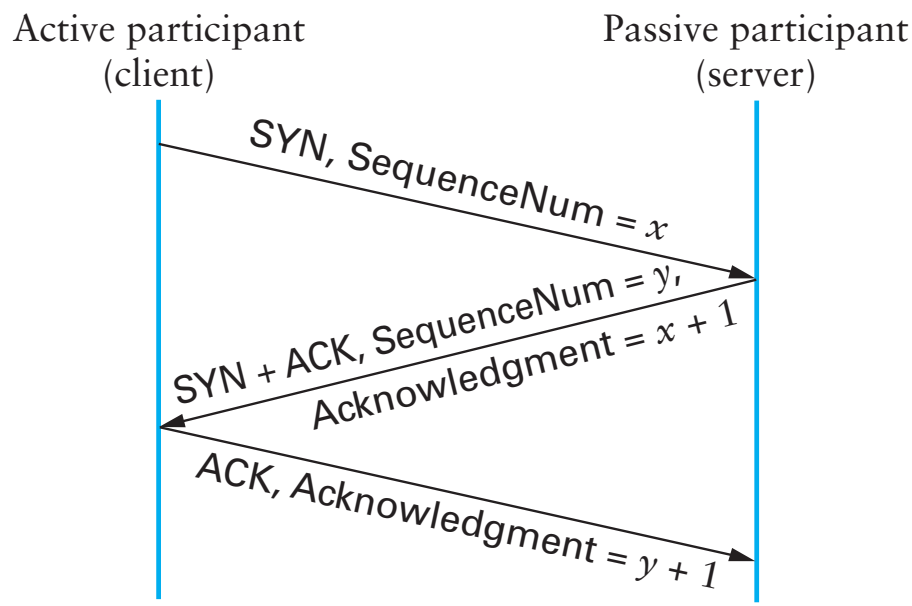
TCP Flags

- URG – urgent data present
- ACK – ack no. valid (all but first segment)
- PSH – push data up to application immediately
- RST – reset connection
- SYN – “synchronize” establishes connection
- FIN – close connection

A TCP Connection (no data)

```
orchard.48150 > essex.discard:
        S 1871560457:1871560457(0) win 16384
essex.discard > orchard.48150:
        S 3249357518:3249357518(0) ack 1871560458 win 17376
orchard.48150 > essex.discard: . ack 1 win 17376
orchard.48150 > essex.discard: F 1:1(0) ack 1 win 17376
essex.discard > orchard.48150: . ack 2 win 17376
essex.discard > orchard.48150: F 1:1(0) ack 2 win 17376
orchard.48150 > essex.discard: . ack 2 win 17375
```

Connection establishment

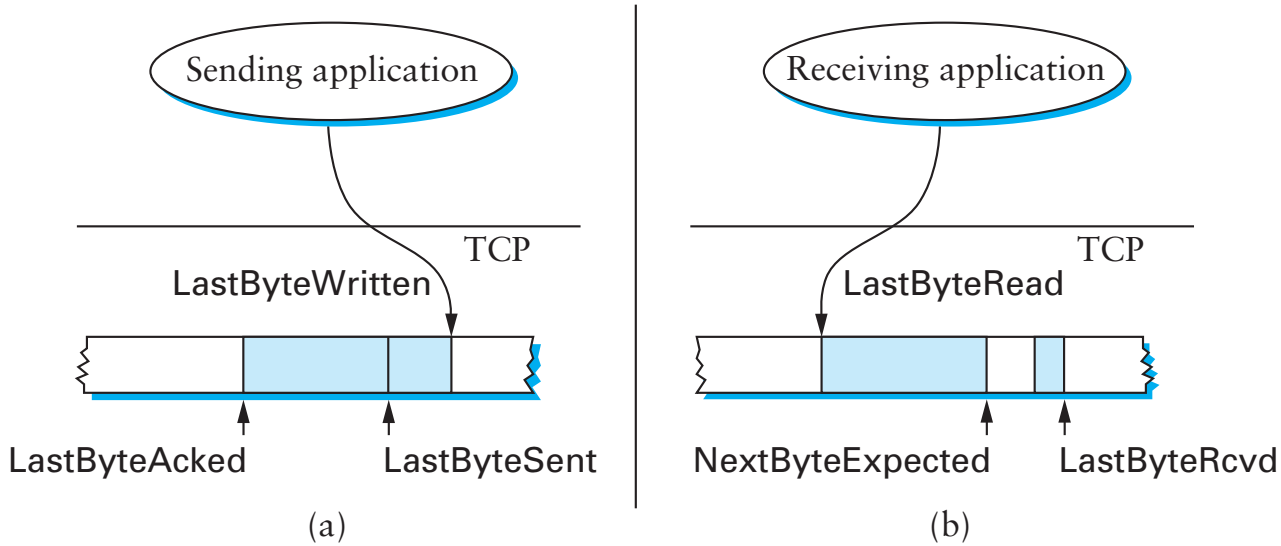


- **Need SYN packet in each direction**
  - Typically second SYN also acknowledges first
  - Supports "simultaneous open," seldom used in practice
- **If no program listening: server sends RST**
- **If server backlog exceeded: ignore SYN**
- **If no SYN-ACK received: retry, timeout**

Sending data

- **Data sent in MSS-sized segments**
  - Chosen to avoid fragmentation (e.g., 1460 on ethernet LAN)
  - Write of 8K might use 6 segments—PSH set on last one
  - PSH avoids unnecessary context switches on receiver
- **Sender's OS can delay sends to get full segments**
  - Nagle algorithm: Only one unacknowledged short segment
  - TCP_NODELAY option avoids this behavior
- **Segments may arrive out of order**
  - Sequence number used to reassemble in order
- **Window achieves flow control**
  - If window 0 and sender's buffer full, write will block or return EAGAIN

Sliding window



- **Used to guarantee reliable & in-order delivery**
- **Also used for flow control**
  - Instead of fixed window size, receiver sends AdvertisedWindow

A TCP connection (3 byte echo)

```
orchard.38497 > essex.echo:
        S 1968414760:1968414760(0) win 16384
essex.echo > orchard.38497:
        S 3349542637:3349542637(0) ack 1968414761 win 17376
orchard.38497 > essex.echo: . ack 1 win 17376
orchard.38497 > essex.echo: P 1:4(3) ack 1 win 17376
essex.echo > orchard.38497: . ack 4 win 17376
essex.echo > orchard.38497: P 1:4(3) ack 4 win 17376
orchard.38497 > essex.echo: . ack 4 win 17376
orchard.38497 > essex.echo: F 4:4(0) ack 4 win 17376
essex.echo > orchard.38497: . ack 5 win 17376
essex.echo > orchard.38497: F 4:4(0) ack 5 win 17376
orchard.38497 > essex.echo: . ack 5 win 17375
```

Retransmission

- **TCP dynamically estimates round trip time**
- **If segment goes unacknowledged, must retransmit**
- **Use exponential backoff (in case loss from congestion)**
- **After $\sim$10 minutes, give up and reset connection**
- **Many optimizations in TCP**
  - E.g., Don't necessarily halt everything for one lost packet
  - Just reduce window by half, then slowly augment

DoS attacks

- **In Feb. 2000, Yahoo's router kept crashing**
  - Engineers had problems with it before, but this was worse
  - Turned out they were being flooded with ICMP echo replies
  - Many DDoS attacks followed against high-profile sites
- **Basic Denial of Service attack**
  - Overload a server or network with too many packets
  - Mamize cost of each packet to server in CPU and memory
- **Distributed DoS (DDos) particularly effective:**
  - Penetrate many machines in semi-automatic fashion
  - Make hosts into "zombies" that will attack on command
  - Later start simultaneous widespread attacks on a victim

Smurf attack

- **Yahoo attack was smurf attack**
  - Penetrated hosts on well-connected networks
  - Flooded LAN with broadcast pings “from” yahoo
  - Every host on LAN then replied to Yahoo
  - Attack was *amplified* through uncompromised hosts
- **Can tolerate above by filtering packets**
  - Packets all ICMP echo replies from particular addresses
  - Attack still had to be traced to stop waste
  - But attack packets could be distinguished from most legitimate traffic

The SYN-bomb attack

- **TCP handshake:**
  - $C \rightarrow S$: SYN, $S \rightarrow C$: SYN-ACK, $C \rightarrow S$: ACK
- **How to implement:**
  - Server inserts connection state in a table
  - Waits for 3rd packet (times out after a minute)
  - Compares each new ack packet to existing connections
- **OS can't handle arbitrary # partial connections**
- **Attack: Send SYN packets from bogus addresses**
  - SYN-ACKs will go off into the void
  - Server's tables fill up, stops accepting connections
  - A few hundred pkts/sec completely disables most servers

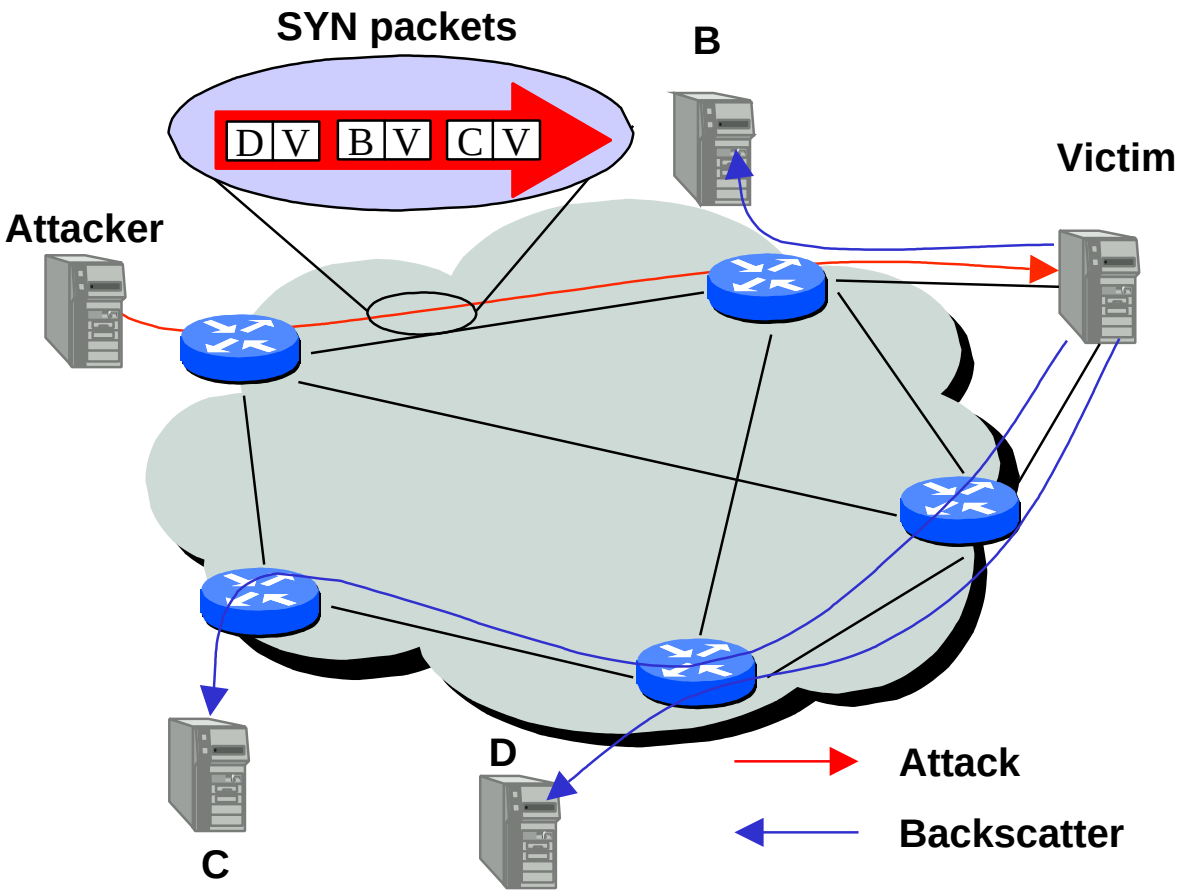
Other attacks

- **IP Fragment flooding**
  - Kernel must keep IP fragments around for partial packets
  - Flood it with bogus fragments, as with TCP SYN bomb
- **UDP echo port 7 replies to all packets**
  - Forge packet from port 7, two hosts echo each other
  - Has been fixed in most implementations
- **Standard flooding attacks**
  - Just flood-ping any site
  - Or bombard DNS server with requests

Making attacks hard to stop

- **Make DoS traffic indistinguishable from legit**
  - SYN-bomb ideal, DNS or any UDP service good
  - Flood-ping at least can be filtered anywhere upstream
- **Make source of attack hard to trace**
  - Victims need to trace attack and pull the plug
  - Can forge source IP address so packet origin not obvious
  - Most DoS tools use a random address for each packet
  - Can also use reflectors–bounce attack through 3rd parties

Backscatter



- **Premise: Many DoS attacks produce backscatter**
  - random IP source address gets reply

Measuring DoS activity

- **Measure backscatter to quantify DoS attacks**
  - If m backscatter packets sent and you monitor n addresses, then $\sim nm/2^{32}$ attack packets were sent.
- **Researchers got lightly-loaded class-A network**
  - Represents $1/256$ of all 32-bit IP addresses
  - Single workstation observed all traffic to class-A net
- **See paper for results detailed…**
  - Observed 12,805 attacks in a week

Limitations of Technique

- **Factors that will cause underestimation**
  - Ingress filtering by ISPs
  - Packet loss
  - Reflector attacks
  - Attack packets that don't cause reply (TCP RST bomb)
- **Non-attack packets could cause overestimation**
- **Non-random source IP addresses could affect results either way**

Coping with denial of service

- **Engineering OSes to tolerate attacks**
  - Reduce state required for embryonic TCP connections
  - Increase size of hash table for protocol control blocks
- **Network monitoring box (schuba et al.)**
  - Passively monitors network
  - Uses heuristics to detect SYN bomb attacks
    (e.g., traffic paterns w. invalid source addresses)
  - Monitor engineered to keep little state
  - Send out forged RST packets to free resources on victim

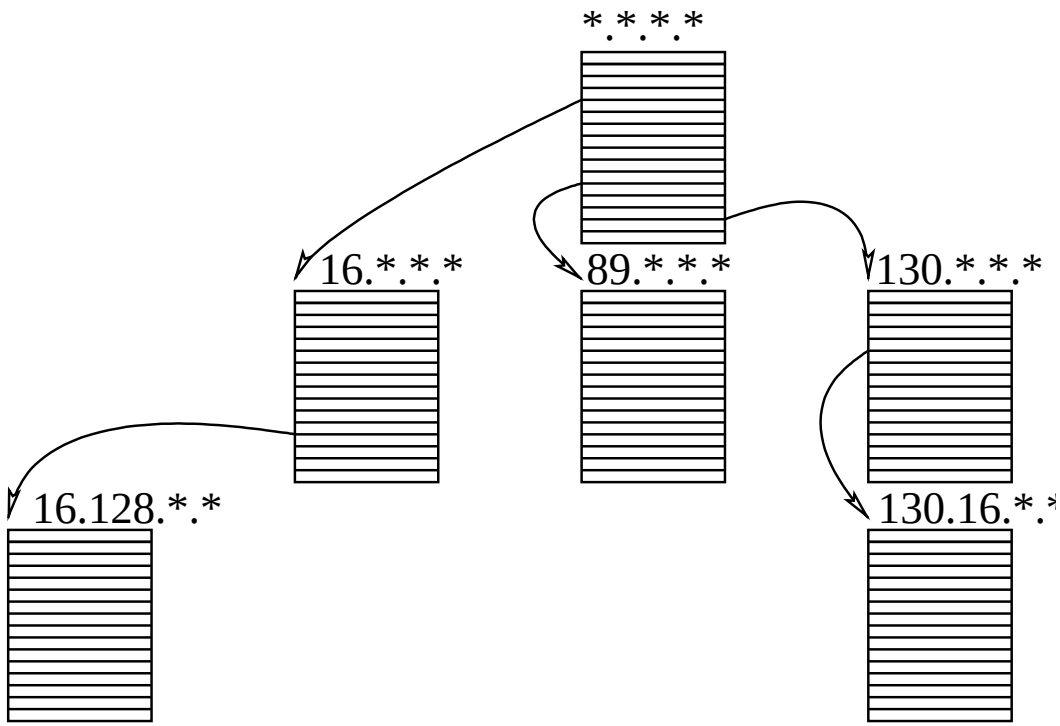
Egress filtering

- **Forged addresses complicate shutting off DoS**
  - Where is flood of packets coming from?
- **Filter forged outgoing packets**
  - Sites should block outgoing packets not from their network
  - ISPs should block packets not from customer's network
- **But still need to detect and shut down attacks**
- **And most attackers can find non-filtered networks anyway**

MULTOPS: Detecting DOS attacks

- **Observation: Many protocols bidirectional**
  - TCP: 0.5–1 ACKs for every data packet
  - DNS, ping: Reply for every request
  - Streaming media (not so easy, but can have heuristics)
- **Substantial imbalance means something is wrong**
  - Many SYNs not getting SYN ACKs (SYN bomb)
  - Many ICMP echo requests not getting replies (ping flood)
- **Attempt to detect problem and filter bad sources**
  - If attackers being egress filtered, will work
  - Can also be run in reverse to detect outgoing attacks
    (E.g., detect if NYU's network is being used for DoS attack)

Multops tree structure



- **Keep aggregate statistics for address prefixes**
  - Subdivide ranges in which an attack is detected
  - Keeps detailed statistics for attackers with limited space
  - Defend against attempts to exhaust memory

Tracing forged packets

- **MULTOPS not useful against forged packets**
  - Don't know which packets to fliter upstream
  - Can't find attacking machine to pull the plug
- **Need to trace attacks back link-by-link**
  - Goal: List of routers, where prefix is path to attacker
- **Many techniques for tracing, with trade-offs:**
  - Management, Bandwidth, Router CPU, Distributed attacks, Post-mortem capability, Preventative vs. Reactive capability

Input debugging

- **Some routers can trace output to input**
  - Develop attack signature to classify bad packets
  - Router tells you which input port they are from
- **Of course, only router administrator can do this**
- **Must continue on to upstream routers, in other realms**
- **Not all routers have this capability**

CenterTrack

- **Problem: ISPs want to trace attacks themselves**
  - Don't want to involve other administrators for each trace
- **CenterTrack: Employ overlay network**
  - Reroute all of victims traffic through an overlay network
  - Can do this by advertising different route with BGP
  - Send all traffic through central tracking router
  - Run input debugging on tracking router

Controlled flooding

- **Problem: Suppose you want to track attack w/o support from network operators**
- **Solution: *controlled flooding* [Burch&Cheswick]**
  - Exploit attacks that cause other hosts to flood you
  - Use knowledge of network to flood along various links
  - Infer source of real attack from interference with your attack
- **Ingenious, but somewhat evil (exploiting hosts)**

ICMP traceback

- **Goal: Let people trace attacks less destructively**
- **Have routers send tracing traffic**
  - Each router randomly chooses 1 in 20,000 packets to trace
  - Sends special ICMP traceback packet including packet and link that it came from
  - Victim can trace attack back from these packets
- **Unfortunatly, hard to implement**
  - Not all routers know input link when processing packet

ICMP traceback disadvantages

- **ICMP traffic sometimes differentiated from TCP**
  - More willing to drop them when under attack
- **Attacker can flood with forged traceback packets**
  - People will filter traceback packets to survive
  - How to tell real packets from forged ones?
- **Incremental deployment makes tracing hard**
  - Can't line up input link to previous node if previous node isn't generating tracebacks

Packet marking

- **Put tracing information in packets themselves**
- **Node append: The simplest solution**
  - Each router appends its address to every packet
  - Can get attack path from any packet
- **Problem: No room in packets**
  - E.g., with MTU discovery, TCP sends maximum sized segments
  - Would need to fragment, terrible overhead

Node sampling

- **Reserve a single fixed-size node field in header**
  - Just enough to hold IP address of one router—32 bits
- **Routers stamp their addr. in field w. probability p**
- **Eventually victim will get stamps from whole path**
  - Get stamp from d hops upstream with prob. $p(1-p)^{d-1}$
  - Can infer number of hops d from # of pkts. w. stamps
  - If $p > 0.5$, attacker cannot fake closer routers
- **Limitations**
  - Need many packets to trace away nodes. With $p = 0.5$, $\sim 300,000$ pkts. needed for 95% confidence in router order
  - Even 32 bits hard to find in all packets
  - Hard to separate paths from multiple attackers

Edge sampling

- **Add three fields to each packet: start, end, distance**
- **Router at address A marks packets as follows:**
  - With prob. p: start $\leftarrow A$, distance = 0
  - Else: distance++. If distance was 0, end $\leftarrow A$.
- **To reconstruct path, victim makes graph**
  - Starts with own address, inserts edge for each packet
  - Eliminate edges (start,end,d) if d not distance in graph
- **Works well with multiple attackers**
- **Incremental deployment works well, too**
  - An edge is closest two routers implementing system
- **Still requires non-existent space in IP headers**

Compressed edge sampling [Savage et al.]

- **Save a factor of two by XORing start & end**
  - Packet with $d = 0$ contains address of first router
  - XOR that with address in pkt with $d = 1$ to get next hop, etc.
- **Put only the fragment of an address in each packet**
- **Use checksum to add redundancy**
  - 32-bit checksum of IP address interleaved with address bits
  - Try all possible fragment reconstructions
  - Discarded ones in which checksum does not work out

Implementation

- **Use unused fragment ID in non-fragments**
- **You get 16 bits. Allocate as follows:**
  - 3 bit offset (which 1/8 of address is this)
  - 5 bit distance (32 hops is generally enough for internet)
  - 8 bit edge fragment
- **Distance aligned with TTL for checksum**
  - Makes implementation efficient—no change in IP checksum
- **Issue of fragmentation (though $< 0.25\%$ of traffic)**
  - Upstream fragments: If marked, frag IDs may then differ. So trash pkt & use full edge marking w. low probability
  - Downstream: Can get ugly if IDs reused. Could use DF bit.

Route Propagation

- **Know a smarter router**
  - hosts know local router
  - local routers know site routers
  - site routers know core router
  - core routers know everything
- **Introduce notion of *Autonomous System* (AS)**
- **Two-level route propagation hierarchy**
  - interior gateway protocol (each AS selects its own)
  - exterior gateway protocol (Internet-wide standard)

Autonomous systems

- **Corresponds to an administrative domain**
  - Internet is not a single network
  - ASes reflect organization of the Internet
  - E.g., NYU, large company, etc.
- **Goals:**
  - ASes want to choose their own local routing algorithm
  - ASes want to set policies about non-local routing
- **Each AS assigned unique 16-bit number**

Types of AS

- ***Local traffic* – packets with src or dst in local AS**
- ***Transit traffic* – passes through an AS**
- ***Stub* AS**
  - Connects to only a single other AS
- ***Multihomed* AS**
  - Connects to multiple ASes
  - Carries no transit traffic
- ***Transit* AS**
  - Connects to multiple ASes and carries transit traffic

BGP-4

- **Goal: Share connectivity information across ASes**
  - Don't strive for "optimal" routes—too hard
  - Different ASes may have different notions of cost
  - May have policies that dictate suboptimal routes
- **BGP used by two types of routers:**
  - *edge* routers, connecting organization to world
  - *core* routers, making up backbone
- **Within ASes, can use any routing protocol**
  - But backbones too big for RIP or OSPF
  - So *internal* BGP (IBGP) variant for use within ASes

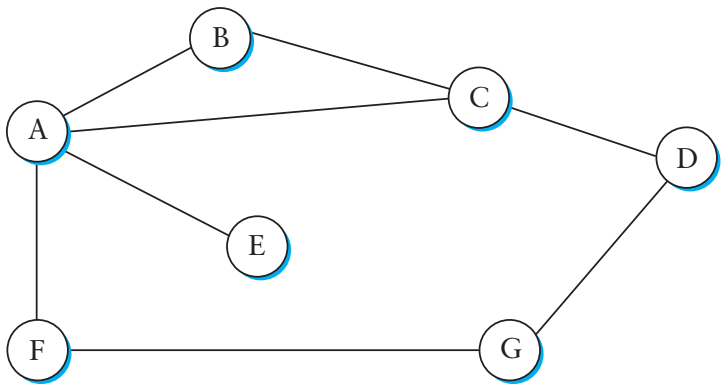
Choice of Routing Algorithm

- **Constraints:**
  - Scaling
  - Autonomy (policy and privacy)
- **What strategy to use?**
  - Distance Vector
  - Link-state

Distance Vector

- **Each node maintains a set of triples**
  - $(Destination, Cost, NextHop)$
- **Exchange updates directly connected neighbors**
  - periodically (on the order of several seconds to minutes)
  - whenever table changes (called triggered update)
- **Each update is a list of pairs:**
  - $(Destination, Cost)$
- **Update local table if receive a "better" route**
  - smaller cost
  - came from next-hop
- **Refresh existing routes; delete if they time out**

Example



Destination	Cost	NextHop
A	1	A
C	1	C
D	2	C
E	2	A
F	2	A
G	3	A

- **B's routing table**

Link State

- **Strategy**
  - Send to all nodes (not just neighbors)
  - Send only information about directly connected links (not entire routing table)

Trade-offs

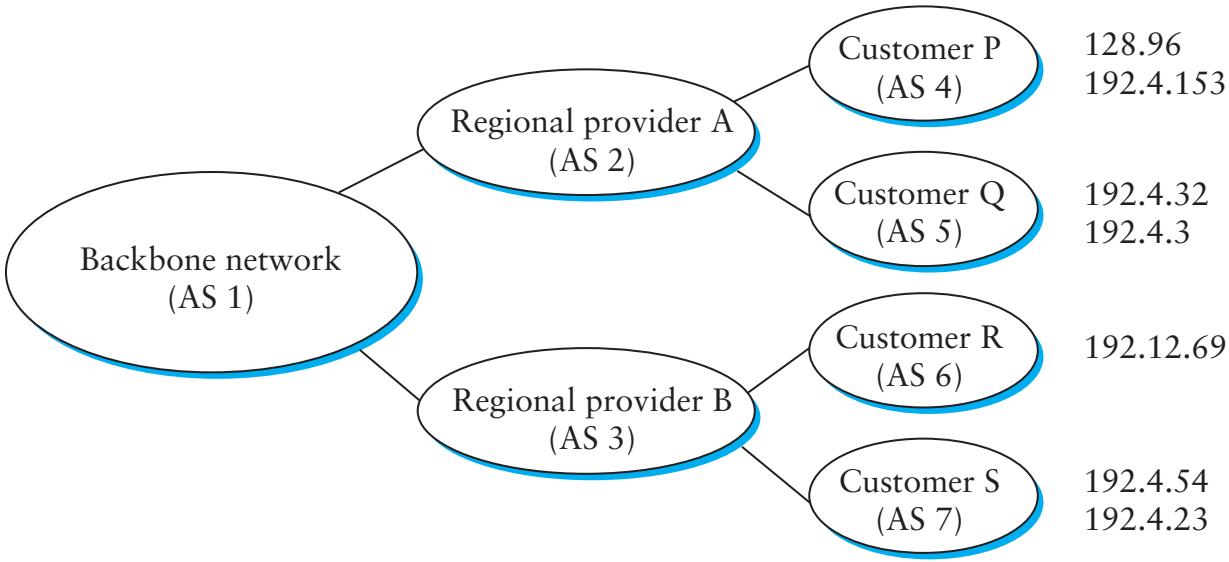
- **Link-state?**
  - Requires sharing of complete network informatin
  - Information exchanges don't scale
  - Can't express policy
- **Distance Vector?**
  - Scales and retains privacy
  - Can't implement policy
  - Can't avoid loops if shortest paths not taken

Path Vector Protocol

- **Distance vector algorithm with extra information**
  - For each route, store the complete path (ASs)
  - No extra computation, just extra storage
- **Advantages:**
  - Can make policy choices based on set of ASs in path
  - Can easily avoid loops
- **In addition, separate *speaker* & *gateway* roles**
  - *speaker* talks BGP protocol to other ASes
  - *gateways* are routers that border other ASes
  - Can have more gateways than speakers
  - Speaker can reach gateways over local network

BGP Example

- **Speaker for AS2 advertises reachability to P and Q**
  - network 128.96, 192.4.153, 192.4.32, and 192.4.3, can be reached directly from AS2



- **Speaker for backbone advertises**
  - networks 128.96, 192.4.153, 192.4.32, and 192.4.3 can be reached along the path (AS1, AS2).
- **Speaker can cancel previously advertised paths**

Basic BGP Messages

- **Open:**
  - Establishes BGP session (uses TCP port #179)
- **Notification:**
  - Report unusual conditions (message header error, . . .)
- **Update:**
  - Inform neighbor of new routes that become active
  - Inform neighbor of old routes that become inactive
- **Keepalive:**
  - Inform neighbor that connection is still viable

Attributes of BGP routes

- **AS path**
- **Origin**
  - Who originated the announcement?
  - IGP, EGP, or "incomplete" (for static routes)
- **Multi-Exit Discriminator (MED)**
  - How close prefix is to link it is announced on
  - Used if ASes A & B connect at multiple points
- **Local preference**
  - Used in IBGP to select (or give preference to) a particular exit for a particular prefix