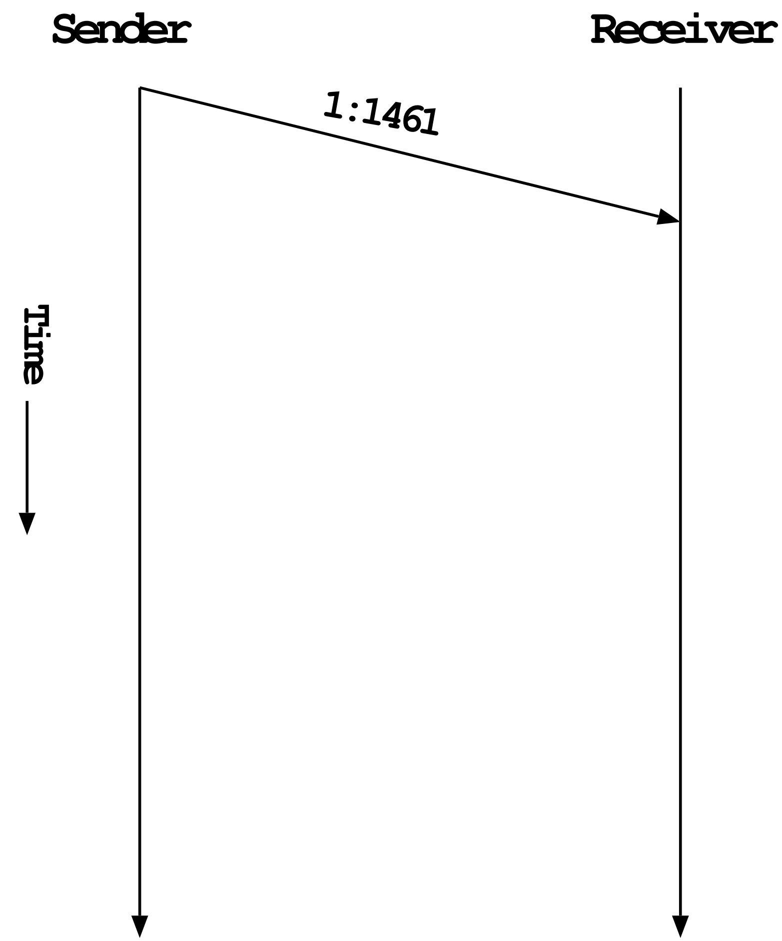
Misbehaving TCP Receivers Can Cause Internet-Wide Congestion Collapse

Rob Sherwood   Bobby Bhattacharjee   Ryan Braud

University of Maryland   UCSD

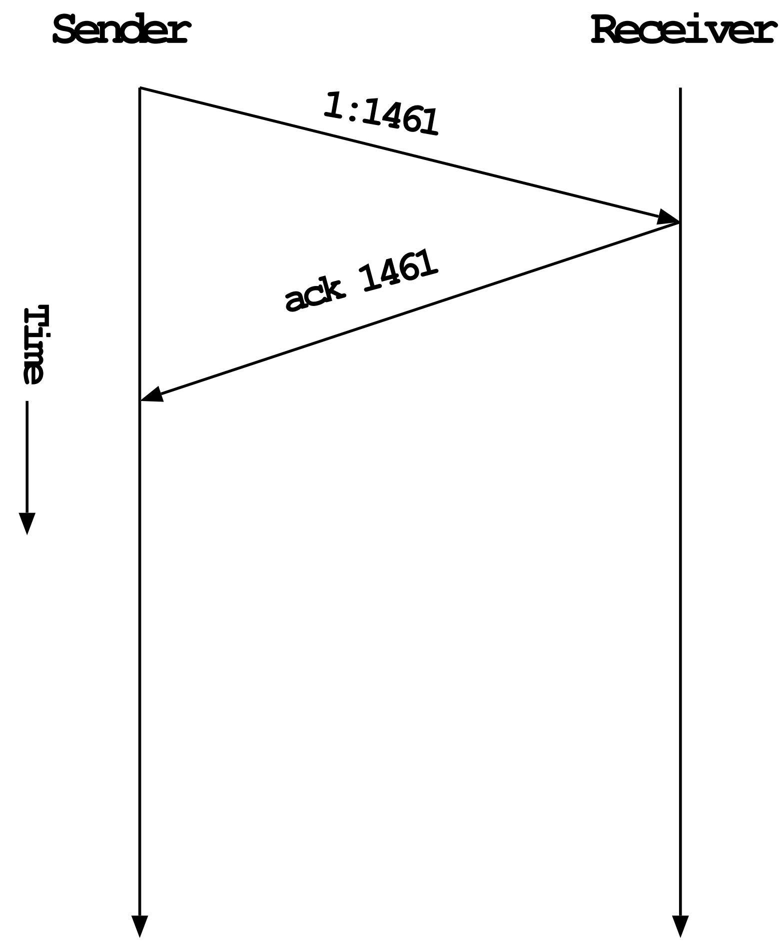
DoS Using TCP Congestion Control

- Sender transmits packet



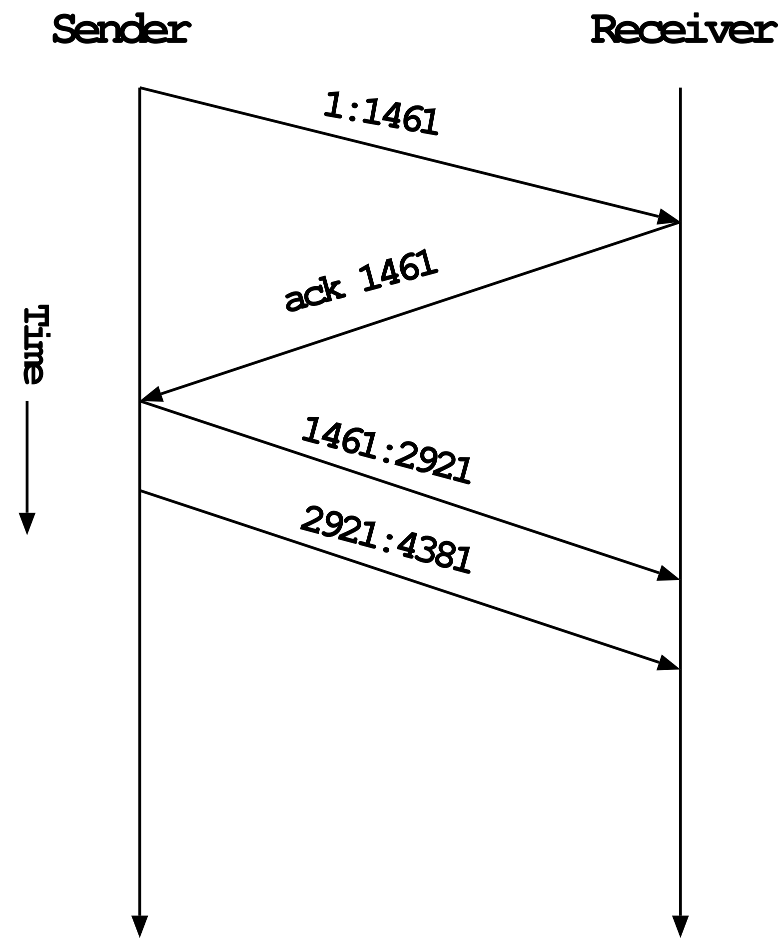
DoS Using TCP Congestion Control

- Sender transmits packet
- Receiver ACKs



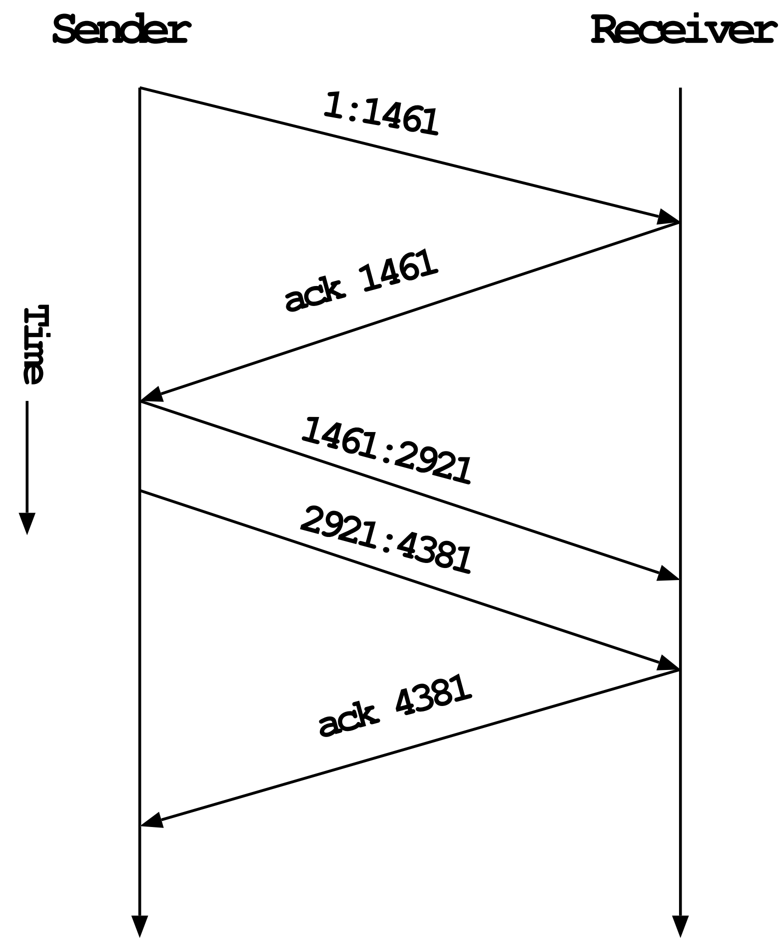
DoS Using TCP Congestion Control

- Sender transmits packet
- Receiver ACKs
- Congestion window grows as long as ACKs continue



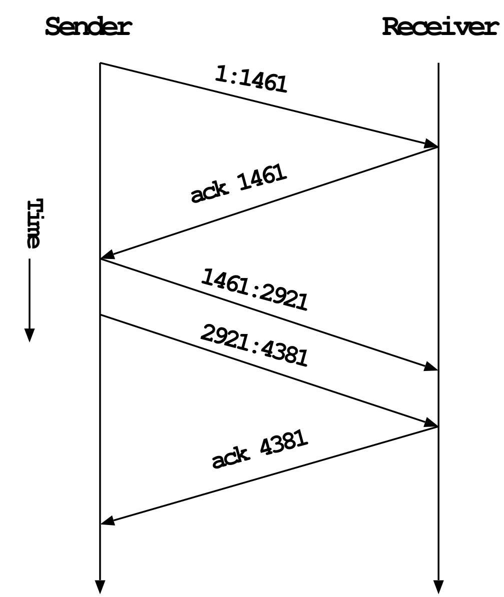
DoS Using TCP Congestion Control

- Sender transmits packet
- Receiver ACKs
- Congestion window grows as long as ACKs continue
- ACKs are cumulative



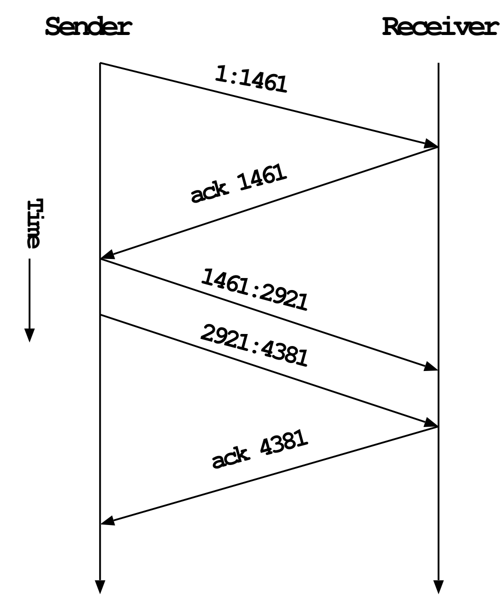
DoS Using TCP Congestion Control

- From ACKs, sender infers:
  - Packet Loss



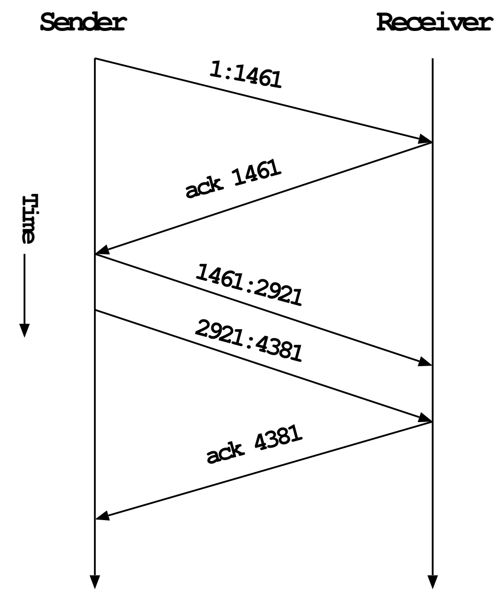
DoS Using TCP Congestion Control

- From ACKs, sender infers:
  - Packet Loss
  - When to send
    - Data is buffered



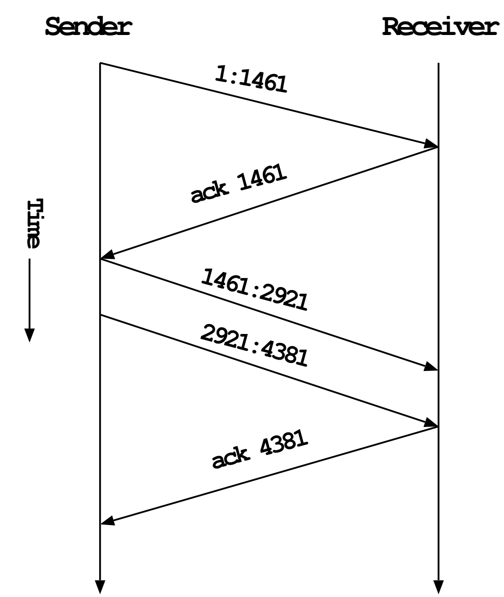
DoS Using TCP Congestion Control

- From ACKs,
  sender infers:
  - Packet Loss
  - When to send
    - Data is buffered
  - How much to send
    - ACKs increase cwnd



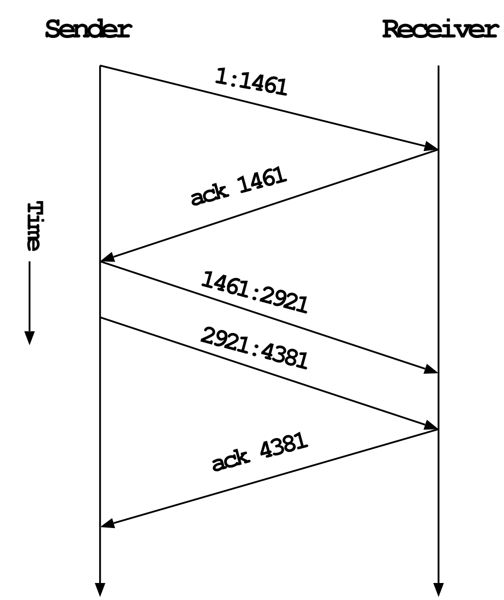
DoS Using TCP Congestion Control

- From ACKs, sender infers:
  - Packet Loss
  - When to send
    - Data is buffered
  - How much to send
    - ACKs increase cwnd
  - When to time out
    - RTT estimation



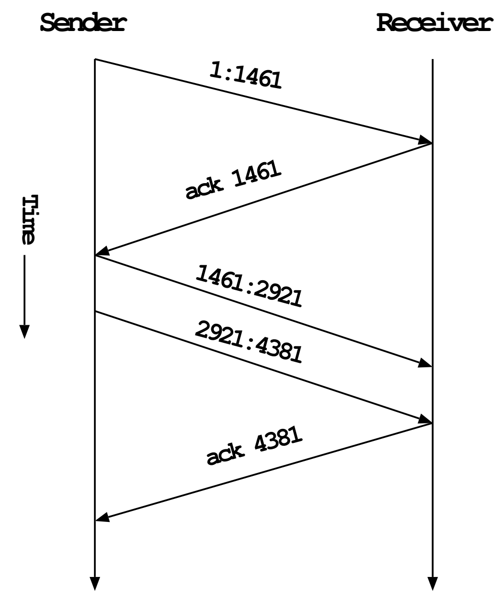
DoS Using TCP Congestion Control

- From ACKs, sender infers:
  - Packet Loss
  - When to send
    - Data is buffered
  - How much to send
    - ACKs increase cwnd
  - When to time out
    - RTT estimation
- Assumes honest feedback



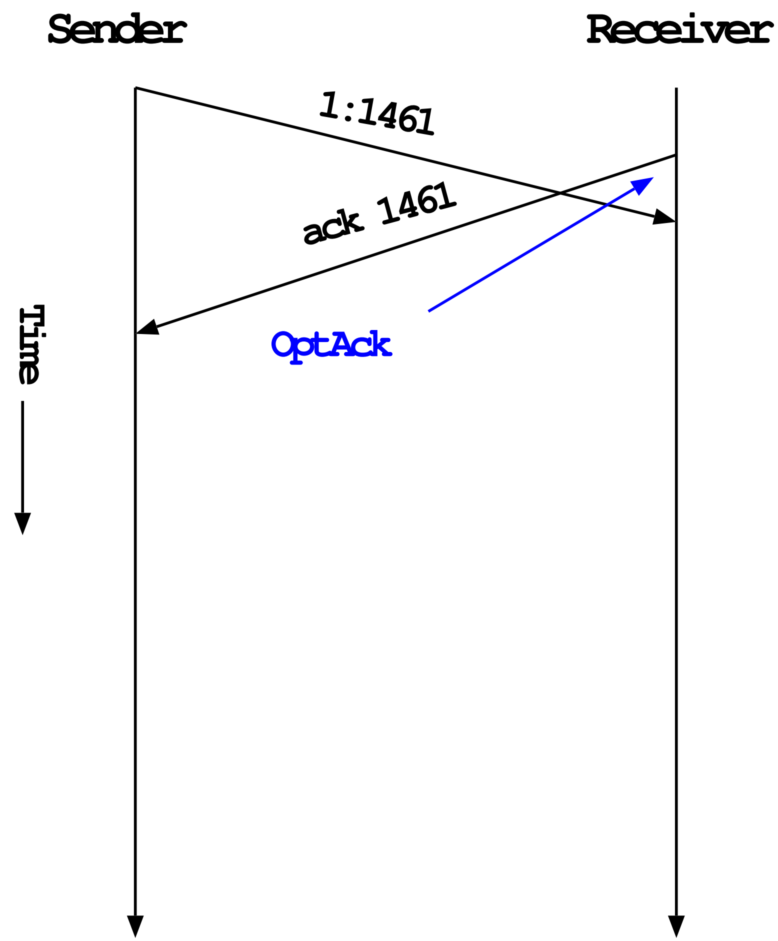
DoS Using TCP Congestion Control

- From ACKs, sender infers:
  - Packet Loss
  - When to send
    - Data is buffered
  - How much to send
    - ACKs increase cwnd
  - When to time out
    - RTT estimation
- What if receiver is malicious?



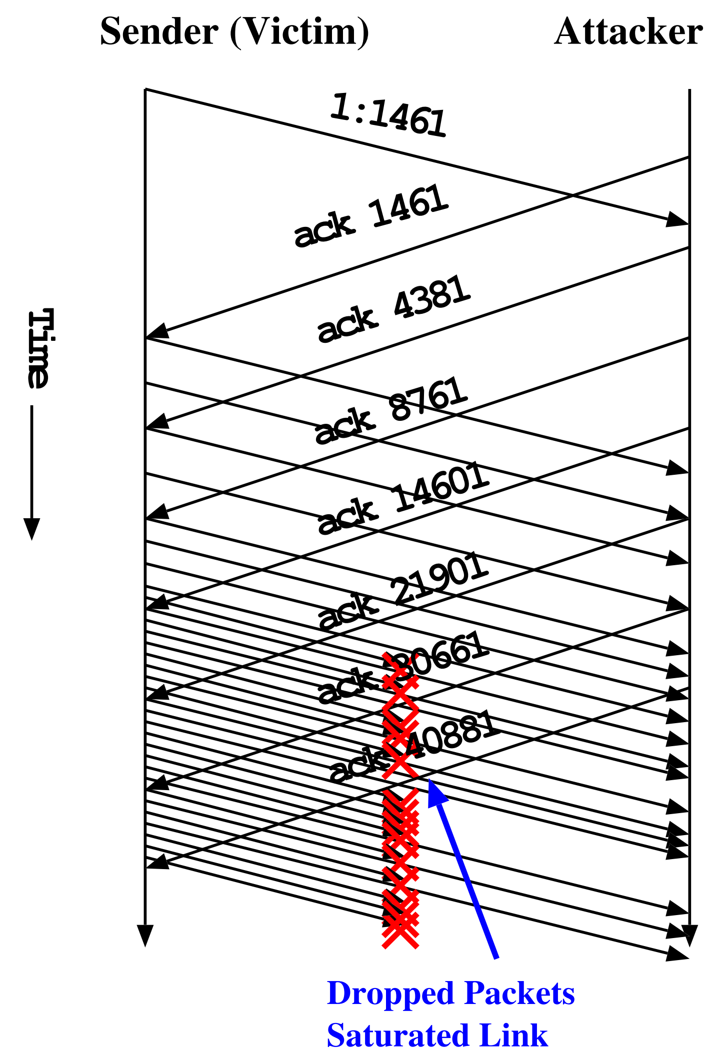
DoS Using TCP Congestion Control

- Misbehaving receiver can send *optimistic* ACKs for packets before they are received
- Faster performance [Savage99]



DoS Using TCP Congestion Control

- Malicious receiver can ACK data that is never received
- Force sender to congest the network (DoS)



Amplification

- Measure of DoS severity
- Amplification = $\frac{\text{traffic generated}}{\text{traffic sent}}$
  - Flooding : x1
  - Spoofed DNS: x4-x10
  - Smurf (broadcast ping): x255

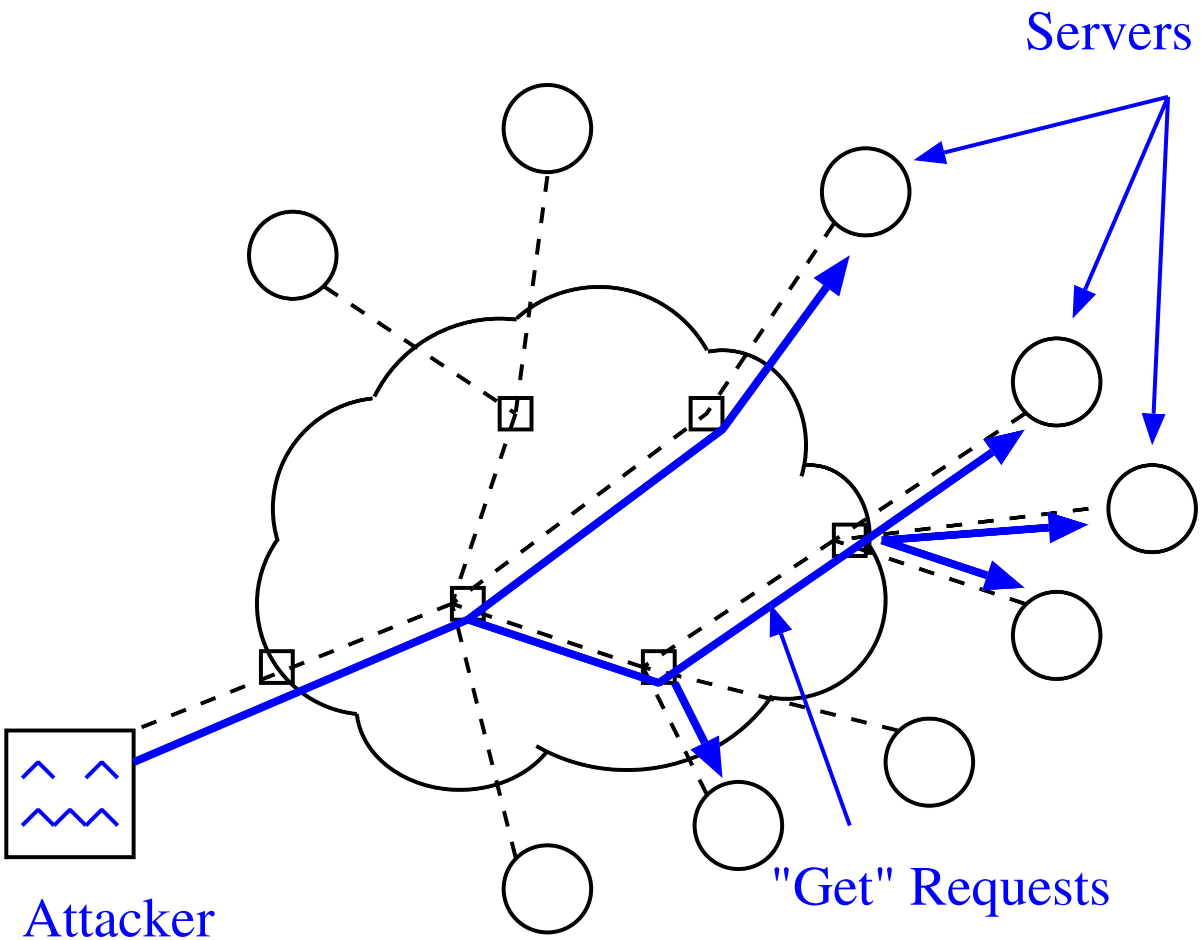
Amplification

- Measure of DoS severity
- Amplification = $\dfrac{\text{traffic generated}}{\text{traffic sent}}$
  - Flooding : x1
  - Spoofed DNS: x4-x10
  - Smurf (broadcast ping): x255
- OptAck: x1683
  - With window scaling: x32×10^6

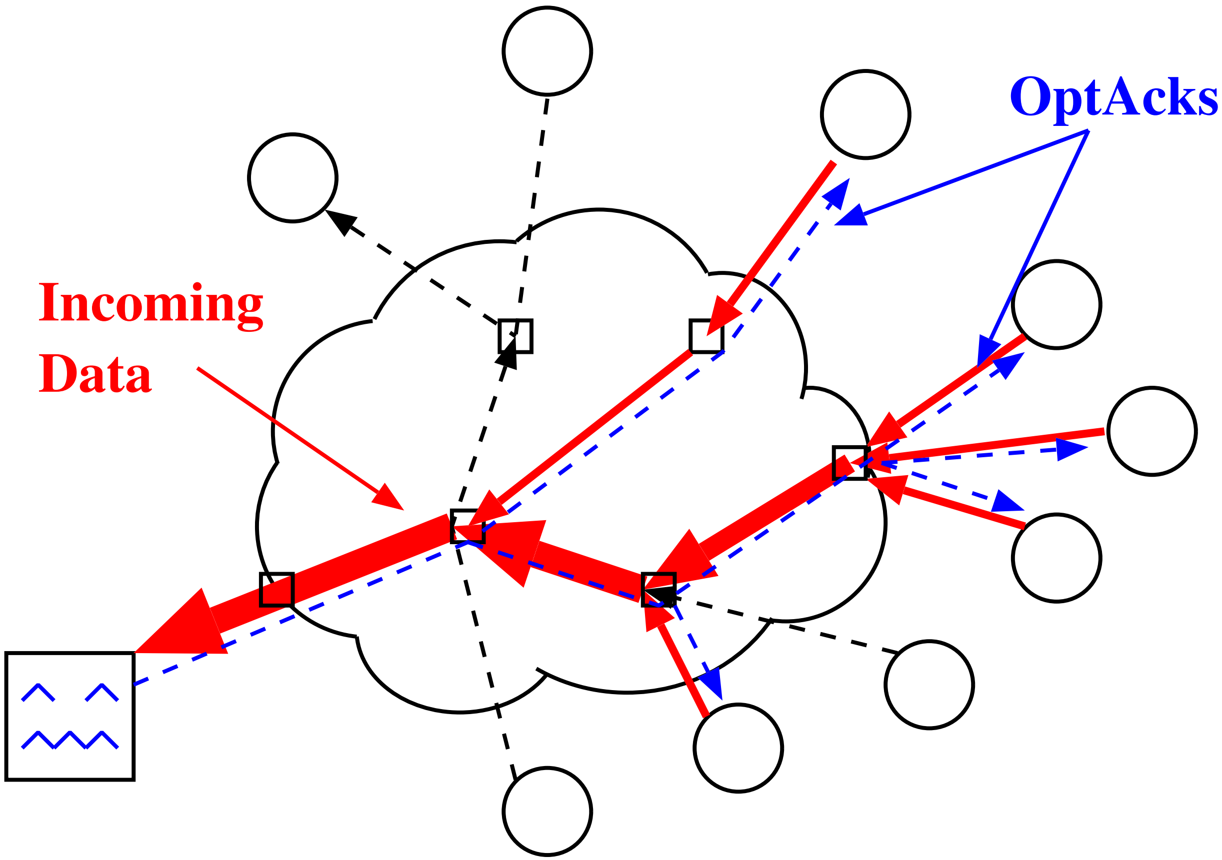
Talk Outline

- Contributions
  - Discuss attack and amplification
  - Implementation techniques
    - Simulated and real world experiments
  - Solution: randomly skipped segments
    - Validate efficiency of implementation
- Conclusions

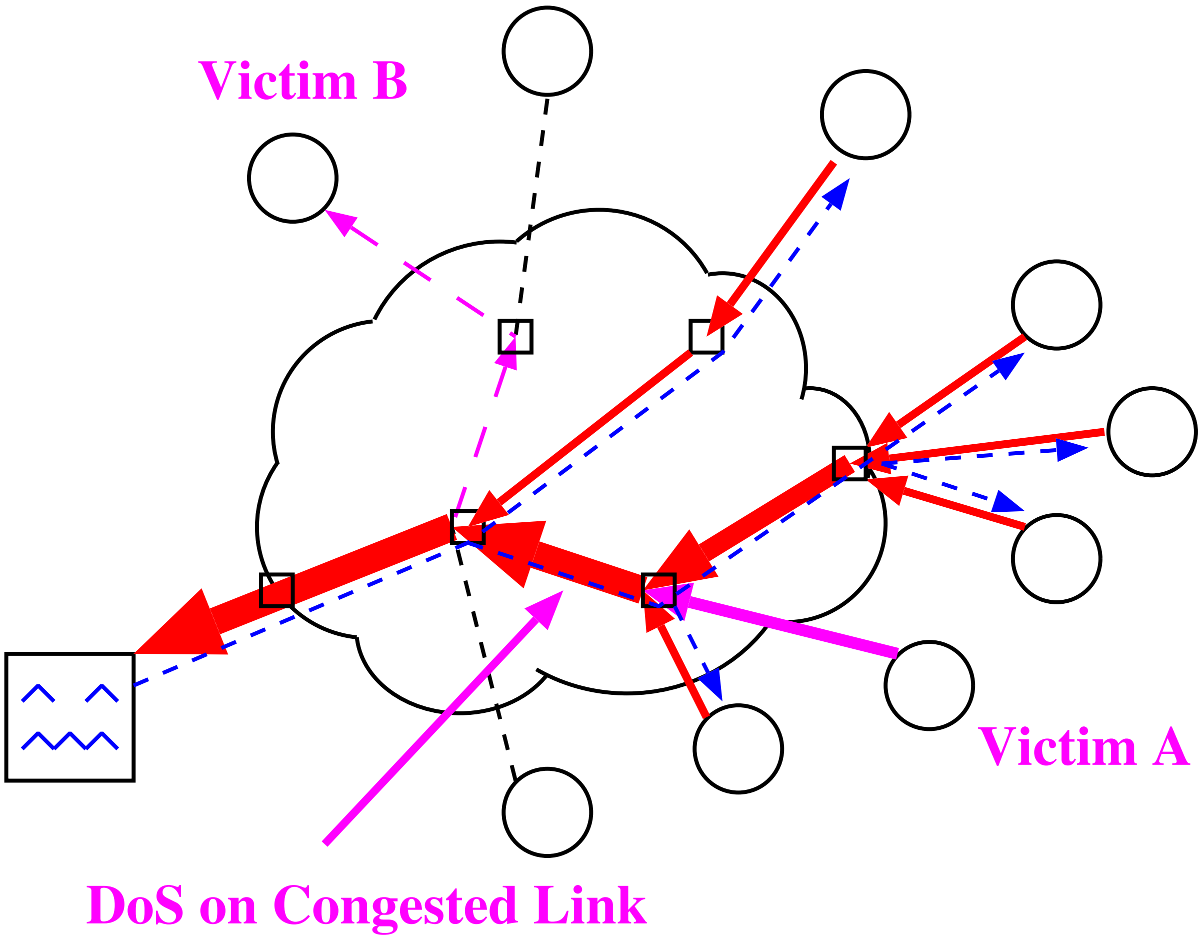
Attack Overview



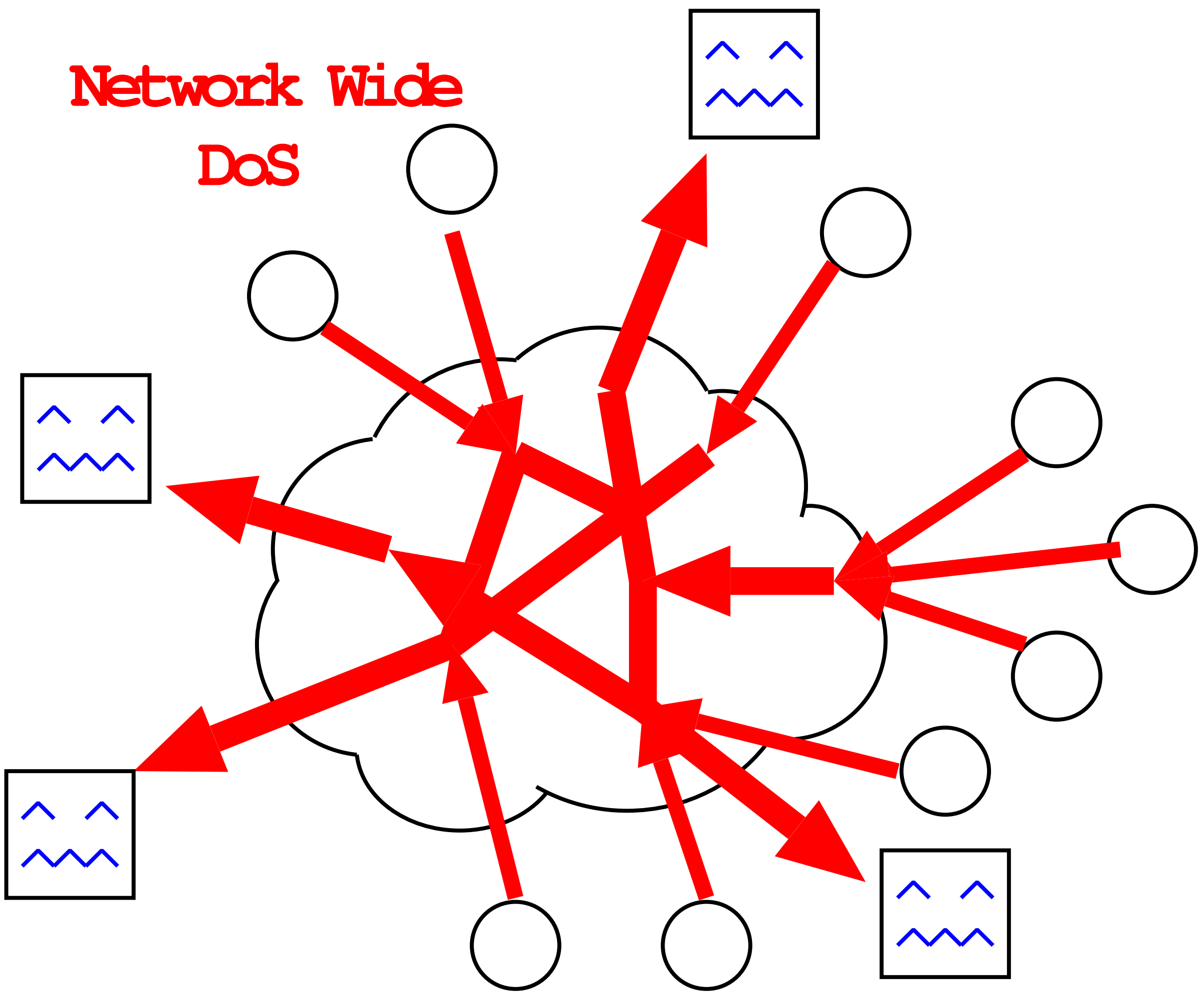
Attack Overview



Attack Overview



Attack Overview



Amplification Factor Analysis

$$\frac{\text{traffic generated}}{\text{traffic sent}}$$

Amplification Factor Analysis

$$\frac{\text{\# packets} \times \text{packet size}}{1 \text{ ACK}}$$

OptAck:

Send 1 ACK $\rightarrow$ Get entire window of packets

Amplification Factor Analysis

$$\frac{\frac{\text{cwnd}}{\text{mss}} \times (40 + \text{mss})}{40}$$

- cwnd: Size of window
- mss: Packet payload size
- 40 bytes: TCP header size

Amplification Factor Analysis

$$\text{cwnd} \times \left[\frac{1}{\text{mss}} + \frac{1}{40}\right]$$

- Typical: cwnd=2^{16}, mss=$1460 \rightarrow$ x1683

Amplification Factor Analysis

$$\text{cwnd} \times \left[\frac{1}{\text{mss}} + \frac{1}{40}\right]$$

- Typical: cwnd=2^{16}, mss=$1460 \rightarrow$ x1683
- [RFC1323] cwnd=$2^{16} \times 2^{\text{wscale}}$
  - wscale=$14 \rightarrow$ cwnd=2^{30}

Amplification Factor Analysis

$$\text{cwnd} \times \left[\frac{1}{\text{mss}} + \frac{1}{40}\right]$$

- Typical: cwnd=2^{16}, mss=$1460 \rightarrow$ x1683
- [RFC1323] cwnd=$2^{16} \times 2^{\text{wscale}}$
  - wscale=$14 \rightarrow$ cwnd=2^{30}
- Smaller mss $\rightarrow$ more amplification

Amplification Factor Analysis

$$\text{cwnd} \times \left[\frac{1}{\text{mss}} + \frac{1}{40}\right]$$

- Typical: cwnd=2^{16}, mss=$1460 \rightarrow$ x1683
- [RFC1323] cwnd=$2^{16} \times 2^{\text{wscale}}$
  - wscale=$14 \rightarrow$ cwnd=2^{30}
- Smaller mss $\rightarrow$ more amplification
- wscale=14, mss=88 $\rightarrow$ x32×10^6

Distributed Attack

- Target CDN's and P2P File transfer nodes

Distributed Attack

- Target CDN’s and P2P File transfer nodes
- Use zombies to attack the Internet’s core

Distributed Attack

- Target CDN's and P2P File transfer nodes
- Use zombies to attack the Internet's core
  - Botnets of 100k+ nodes exist [HONEYNET04]

Distributed Attack

- Target CDN's and P2P File transfer nodes
- Use zombies to attack the Internet's core
  - Botnets of 100k+ nodes exist [HONEYNET04]
- How much traffic before collapse?

Distributed Attack

- Target CDN's and P2P File transfer nodes
- Use zombies to attack the Internet's core
  - Botnets of 100k+ nodes exist [HONEYNET04]
- How much traffic before collapse?
  - Open problem

Distributed Attack

- Target CDN's and P2P File transfer nodes
- Use zombies to attack the Internet's core
  - Botnets of 100k+ nodes exist [HONEYNET04]
- How much traffic before collapse?
  - Open problem
  - Data point: Slammer worm generated 31GB/s at peak [CAIDA03]

Distributed Attack

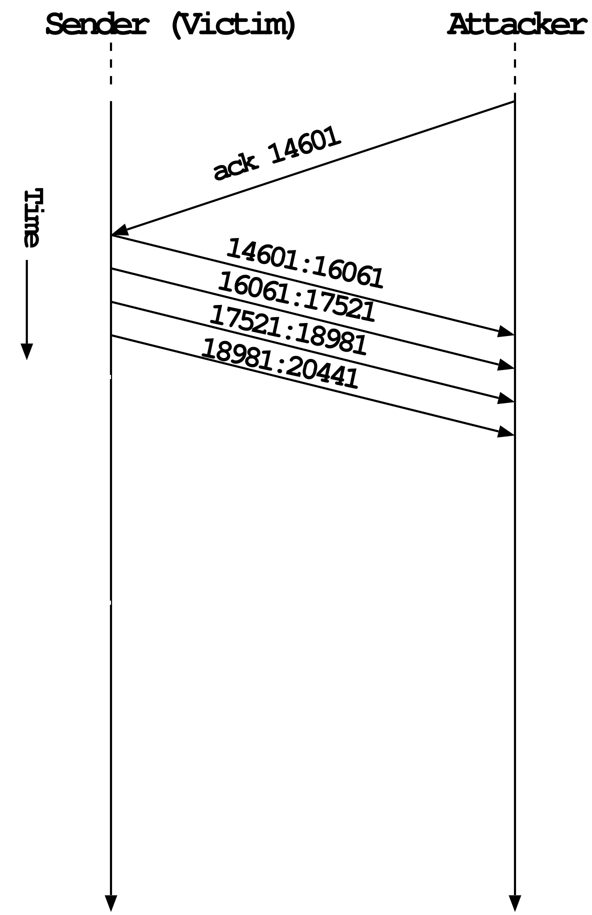
- Target CDN's and P2P File transfer nodes
- Use zombies to attack the Internet's core
  - Botnets of 100k+ nodes exist [HONEYNET04]
- How much traffic before collapse?
  - Open problem
  - Data point: Slammer worm generated 31GB/s at peak [CAIDA03]
    - Equal to traffic generated by 5 OptAck attackers on T3s, wscale=0, mss=1460

Attack Implementation

- Is attack practical?
- How vulnerable are deployed TCP Stacks?
- Discovered effective solution
- We implement the attack
  - approx. 1200 lines of code
- Main challenge: ACK overruns

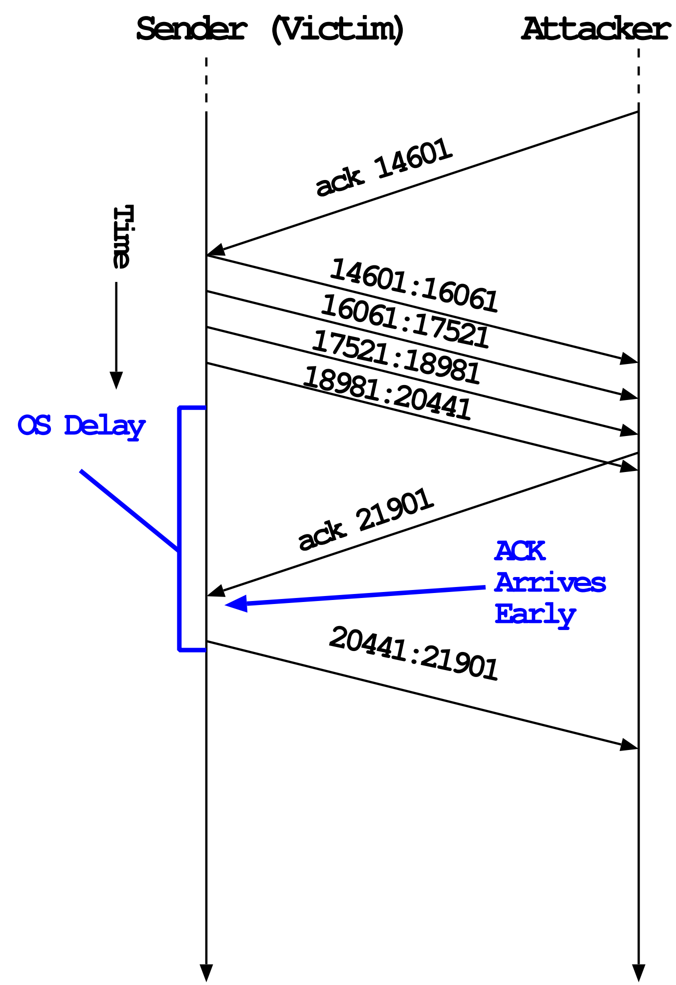
ACK overruns

- Attacker becomes unsynchronized



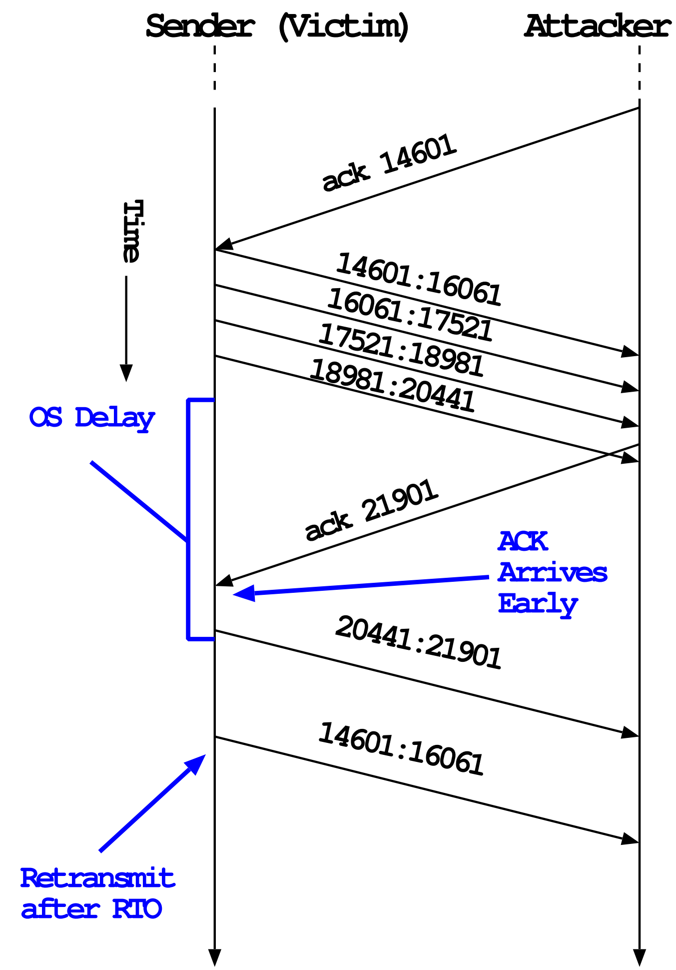
ACK overruns

- Attacker becomes unsynchronized



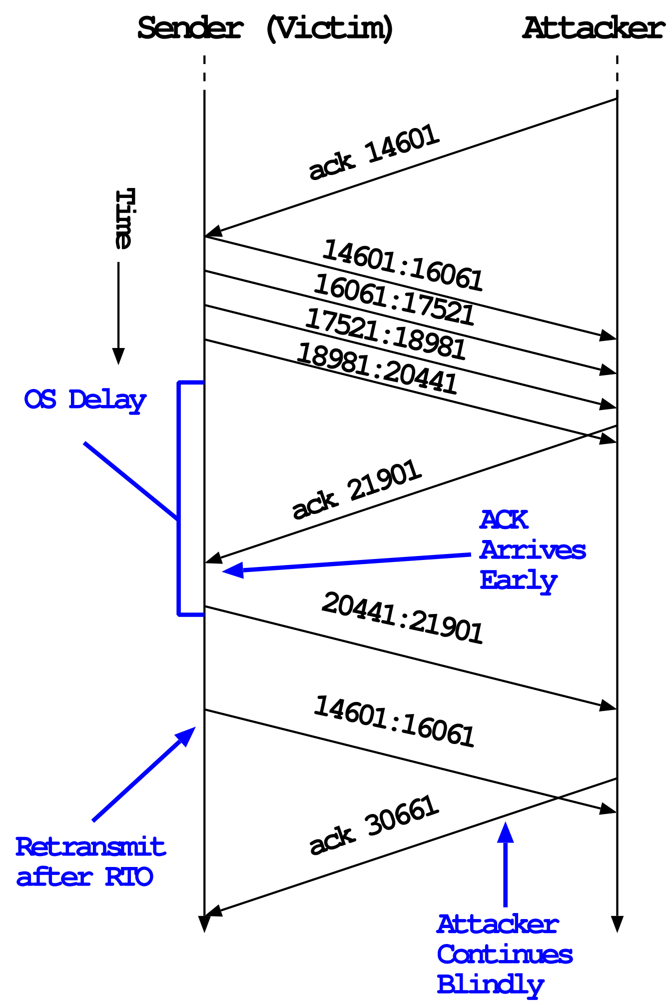
ACK overruns

- Attacker becomes unsynchronized



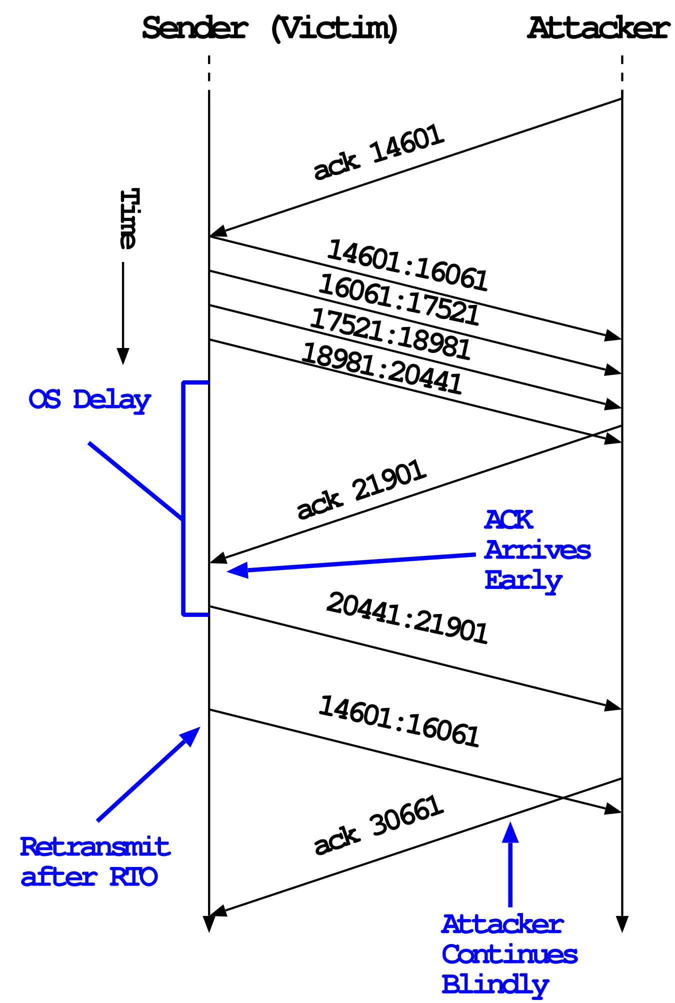
ACK overruns

- Attacker becomes unsynchronized



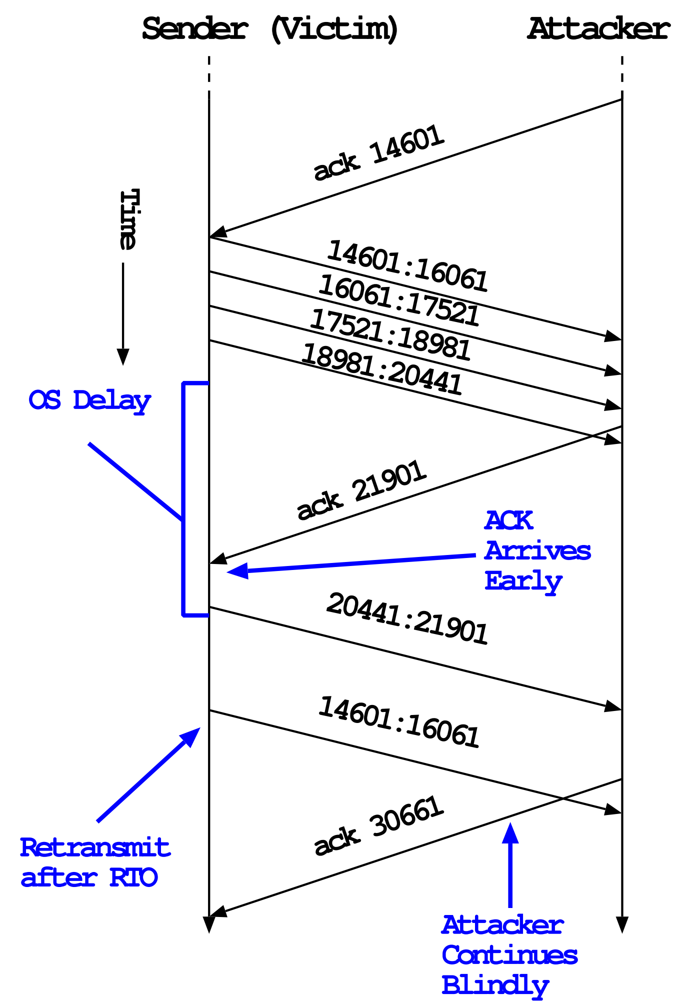
ACK overruns

- Attacker becomes unsynchronized
- Causes of Overrun
  - Server delay
  - ACK compression
  - Dropped ACKs



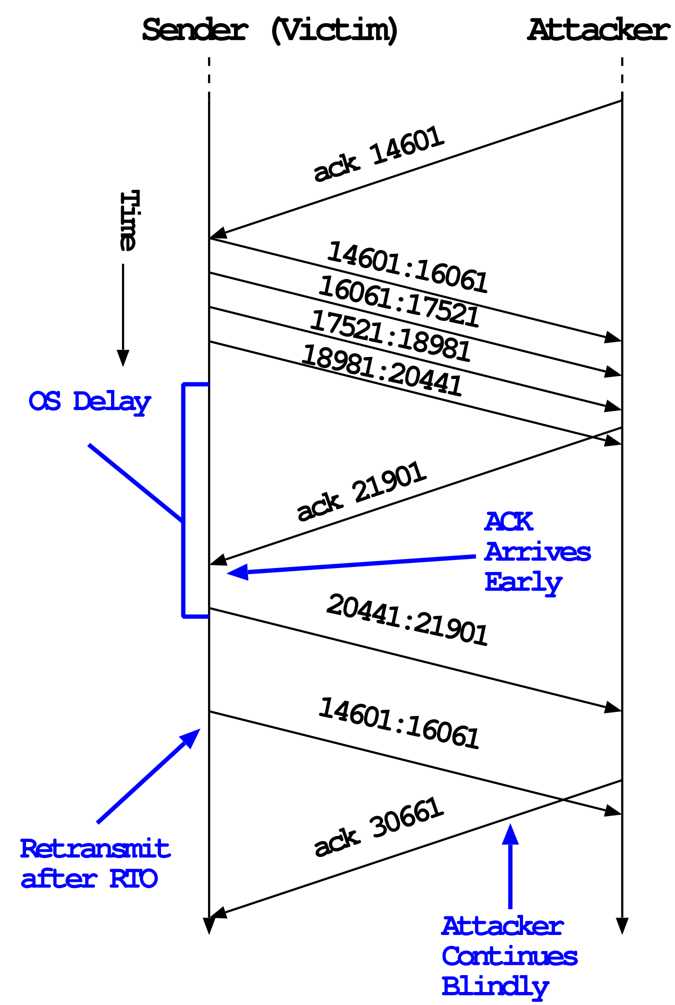
ACK overruns

- Attacker becomes unsynchronized
- Causes of Overrun
  - Server delay
  - ACK compression
  - Dropped ACKs
- [RFC793] says ignore overrun ACKs
  - DoS otherwise



ACK overruns

- Attacker becomes unsynchronized
- Causes of Overrun
  - Server delay
  - ACK compression
  - Dropped ACKs
- [RFC793] says ignore overrun ACKs
  - DoS otherwise
- Overrun avoidance and recovery



Overrun Avoidance and Recovery

- Server delays are short (typically <50ms)

Overrun Avoidance and Recovery

- Server delays are short (typically <50ms)
  - Attacker should target more servers

Overrun Avoidance and Recovery

- Server delays are short (typically <50ms)
  - Attacker should target more servers
- Don't ACK whole window
  - ACK $\frac{1}{2}$ cwnd, twice as often
  - Thwarts ACK compression, dropped ACKs

Overrun Avoidance and Recovery

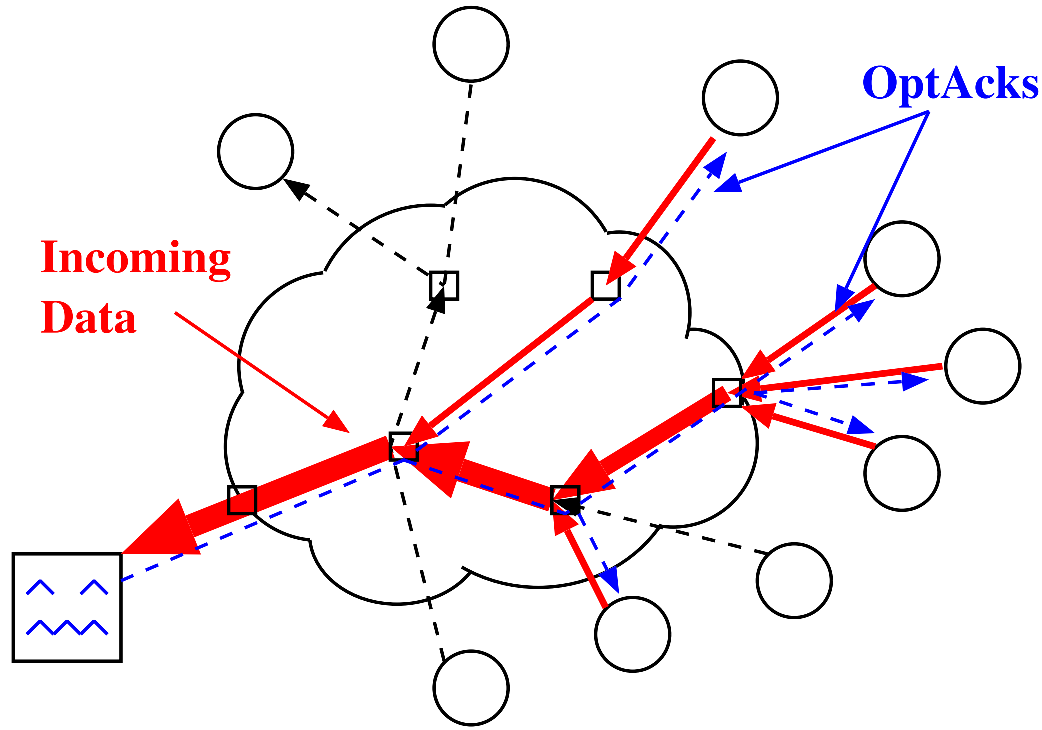
- Server delays are short (typically <50ms)
  - Attacker should target more servers
- Don't ACK whole window
  - ACK $\frac{1}{2}$ cwnd, twice as often
  - Thwarts ACK compression, dropped ACKs
- Don't exceed local attack bandwidth

Overrun Avoidance and Recovery

- Server delays are short (typically <50ms)
  - Attacker should target more servers
- Don't ACK whole window
  - ACK $\frac{1}{2}$ cwnd, twice as often
  - Thwarts ACK compression, dropped ACKs
- Don't exceed local attack bandwidth
- Recovery details in paper

Attack Validation

- Experiment: How much traffic does OptAck generate?



Results - NS Simulations

Connections:

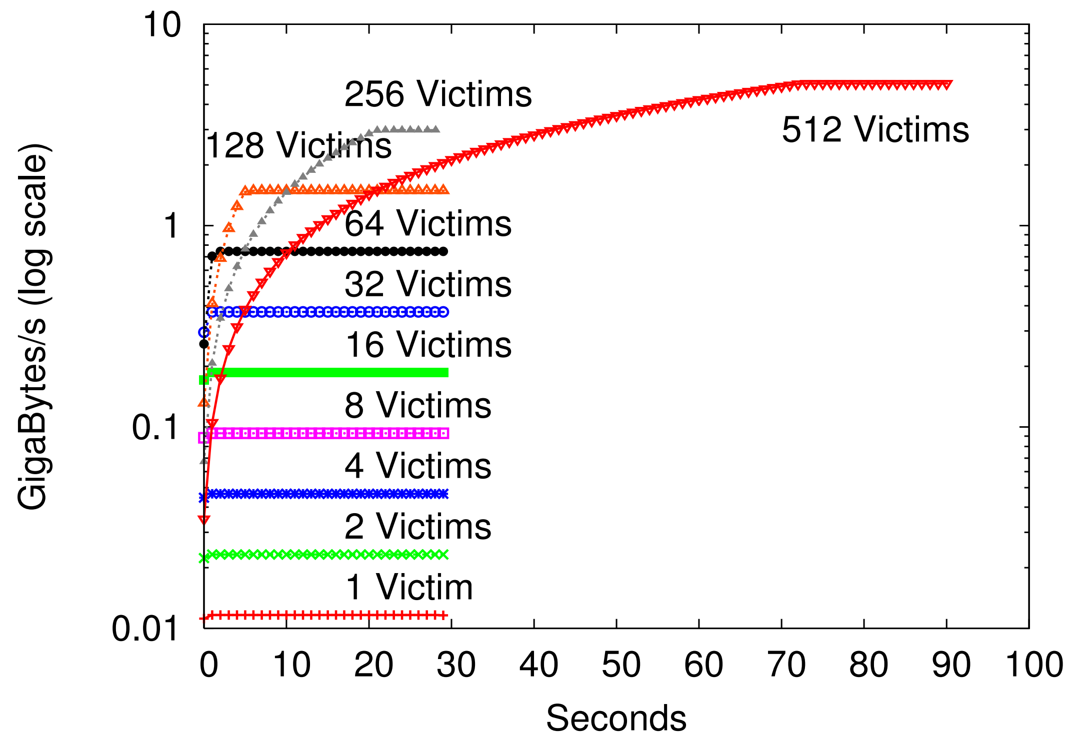
1 Attacker: 1.5Mbps

Victims: 100Mbps

Parameters:

wscale=4

mss=1460



Results - NS Simulations

Connections:

1 Attacker: 1.5Mbps

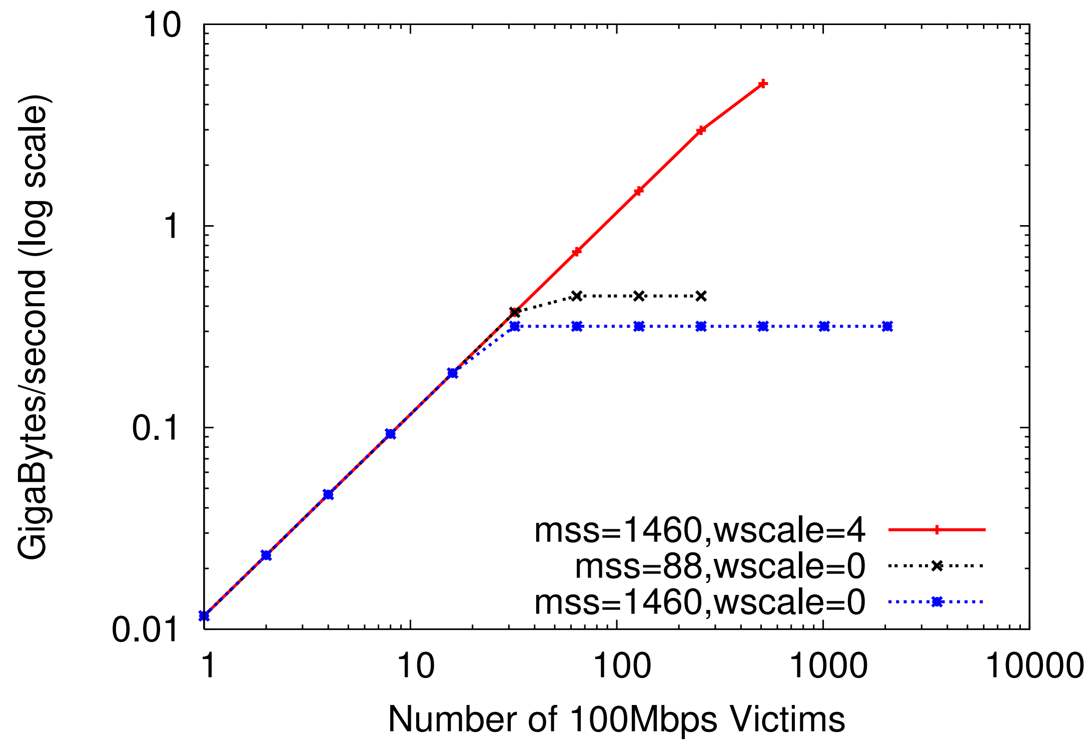
Victims: 100Mbps

Max traffic:

99.9% of predicted

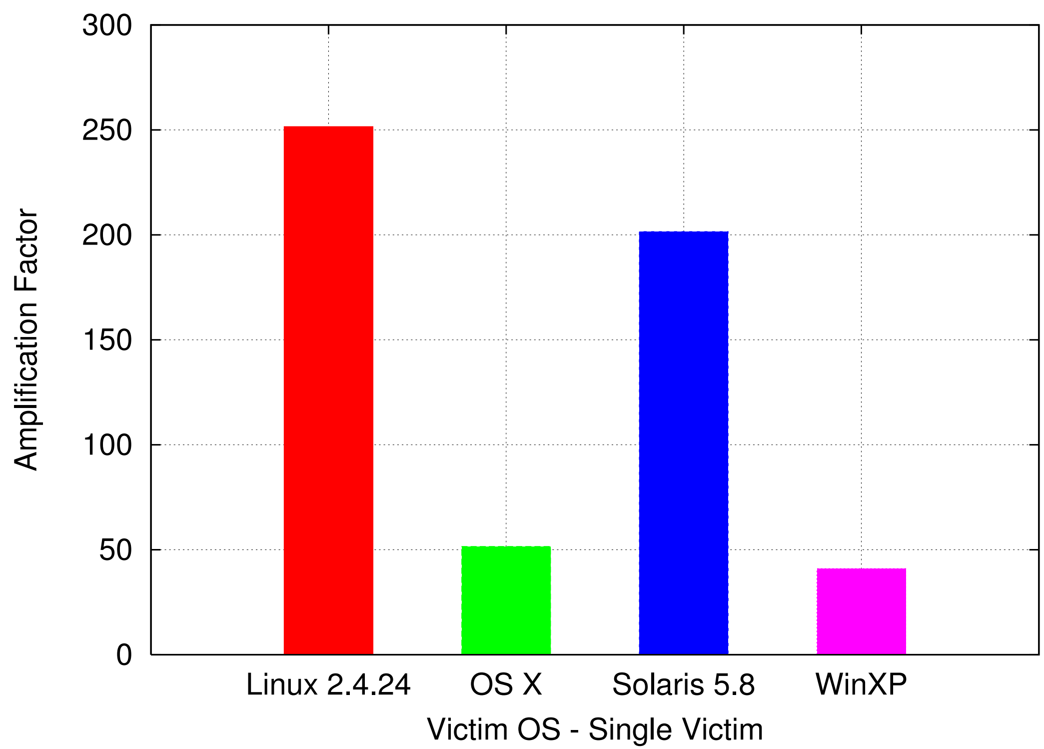
Conclusion:

Model is accurate



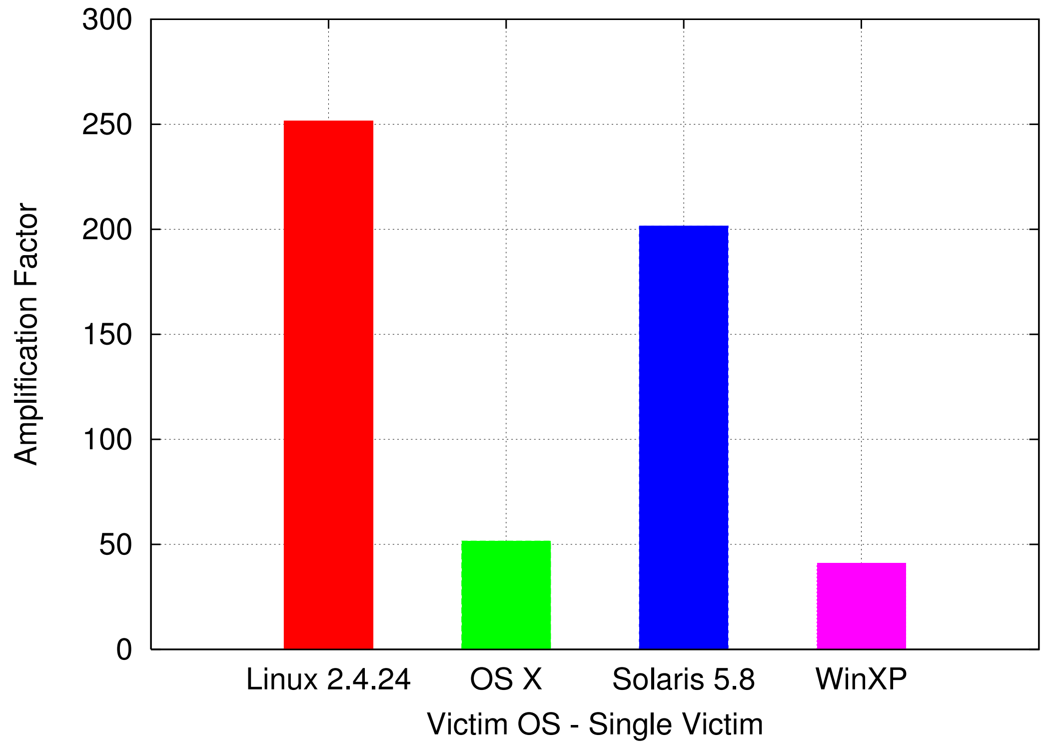
Results - Implementation

- 1 Attacker on LAN w/ 1 victim
- wscale=0, mss=1460
- No multi-victim overrun avoidance



Results - Implementation

- 1 Attacker on LAN w/ 1 victim
- wscale=0, mss=1460
- Conclusion: Works on deployed TCP stacks



Defenses

How to defend against OptAck?

Defenses

How to defend against OptAck?

- Nonce
  - Receiver modification; not deployable

Defenses

How to defend against OptAck?

- Nonce
  - Receiver modification; not deployable
- Rate limiting
  - Target more servers; use more attackers

Defenses

How to defend against OptAck?

- Nonce
  - Receiver modification; not deployable
- Rate limiting
  - Target more servers; use more attackers
- RST on overrun ACK
  - Trivial DoS

Defenses

How to defend against OptAck?

- Nonce
  - Receiver modification; not deployable
- Rate limiting
  - Target more servers; use more attackers
- RST on overrun ACK
  - Trivial DoS
- Network Policing

Defenses

How to defend against OptAck?

- Nonce
  - Receiver modification; not deployable
- Rate limiting
  - Target more servers; use more attackers
- RST on overrun ACK
  - Trivial DoS
- Network Policing
- Insight: attacker cannot tell where in the network a packet is dropped

Defense: Skipped Segments

- Sender should randomly, periodically, skip sending a data segment

Defense: Skipped Segments

- Sender should randomly, periodically, skip sending a data segment
  - Good receiver sends duplicate ACKs; fast retransmit skipped segment
  - Malicious receiver ACKs the hole; RST connection

Defense: Skipped Segments

- Sender should randomly, periodically, skip sending a data segment
  - Good receiver sends duplicate ACKs; fast retransmit skipped segment
  - Malicious receiver ACKs the hole; RST connection
- Cost:
  - Transmission rate unchanged
  - Application receives data 1 RTT later

Defense: Skipped Segments

- Sender should randomly, periodically, skip sending a data segment
  - Good receiver sends duplicate ACKs; fast retransmit skipped segment
  - Malicious receiver ACKs the hole; RST connection
- Cost:
  - Transmission rate unchanged
  - Application receives data 1 RTT later
- Sender only modification: deployable

Defense: Skipped Segment (More)

- Sender should not back off

Defense: Skipped Segment (More)

- Sender should not back off
- Cost of RNG too high per packet
  - Init counter randomly from [*low*,*high*]
  - Decrement counter with each segment
  - counter==0 $\longrightarrow$ skip segment
  - re'init counter randomly

Defense: Skipped Segment (More)

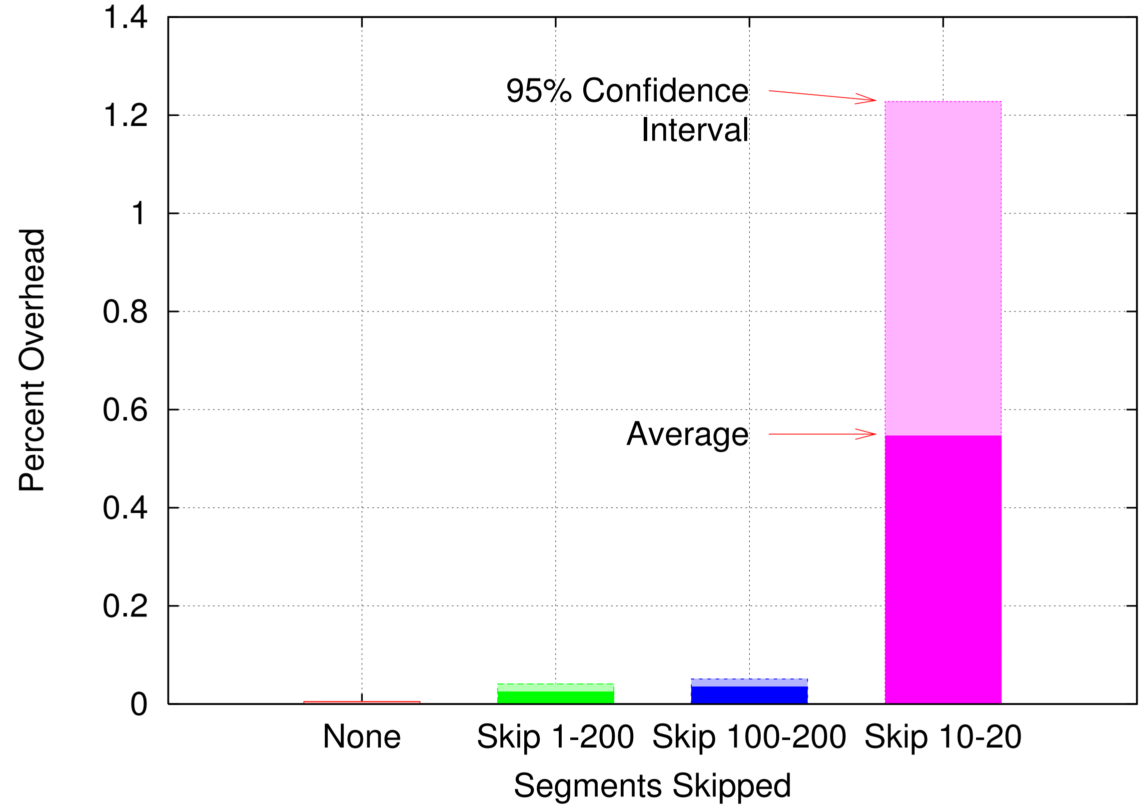
- Sender should not back off
- Cost of RNG too high per packet
  - Init counter randomly from [*low*,*high*]
  - Decrement counter with each segment
  - counter==0 $\longrightarrow$ skip segment
  - re'init counter randomly
- TCP optimized to recover from packet loss

Defense: Skipped Segment (More)

- Sender should not back off
- Cost of RNG too high per packet
  - Init counter randomly from [*low*,*high*]
  - Decrement counter with each segment
  - counter==0 $\rightarrow$ skip segment
  - re'init counter randomly
- TCP optimized to recover from packet loss
- Linux implementation:
  - 30 lines of code
  - 5 bytes per connection

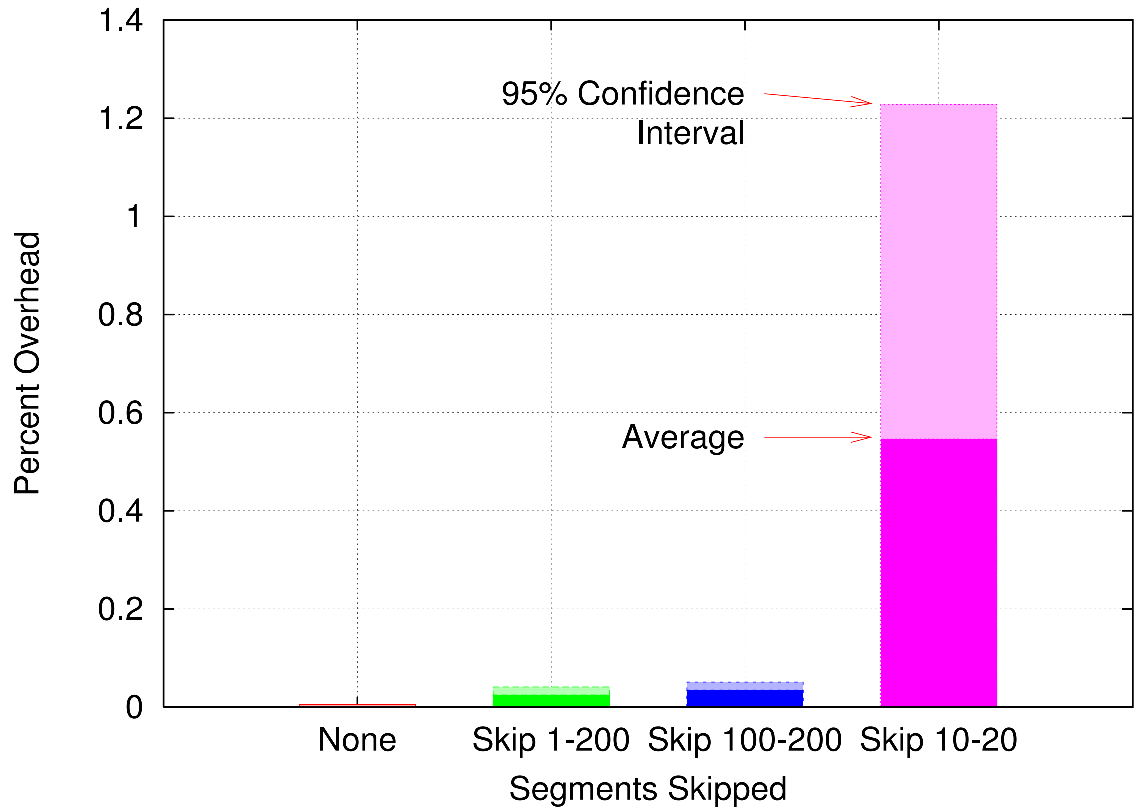
Skipped Segments Efficiency

- Download 100MB file
- No attack
- Vary skip rate



Skipped Segments Efficiency

- Download 100MB file
- No attack
- Vary skip rate
- Conclusion: Overhead is trivial
  - $< 0.1\%$



Related Work

- [Savage99]
  TCP Congestion Control with a Misbehaving Receiver.
  Stefan Savage, Neal Cardwell, David Wetherall, and Tom Anderson.
  CCR 1999.
  - Original optimistic acknowledgment
  - Defense requires receiver side modifications - inhibits deployment

Related Work (more)

- [Azcorra04] (Expired) Draft RFC: DoS vulnerability of TCP by acknowledging not received segments. A. Azcorra, C. Bernardos, and I. Soto. draft-azcorra-tcpm-tcp-blind-ack-dos-01
  - Proposed solutions already discussed in paper

Related Work (more)

- [CAIDA03]: Inside the Slammer Worm: David Moore, Vern Paxson, Stefan Savage, Colleen Shannon, Stuart Staniford, Nicholas Weaver. http://www.caida.org/outreach/papers/2003/sapph
- [HONEYNET04]: Know your Enemy: Tracking Botnets. The Honeynet Project Research Alliance. http://www.honeynet.org/papers/bots.

Conclusion

- OptAck attack is dangerous: x1683
  - x32×10^6 w/ wscaling

Conclusion

- OptAck attack is dangerous: x1683
  - x32×10^6 w/ wscaling
- Deployed TCP Stacks are vulnerable

Conclusion

- OptAck attack is dangerous: x1683
  - x32×10^6 w/ wscaling
- Deployed TCP Stacks are vulnerable
- Skipped segments solution:
  - performs efficiently
  - requires sender only deployment

Conclusion

- OptAck attack is dangerous: x1683
  - x32×10^6 w/ wscaling
- Deployed TCP Stacks are vulnerable
- Skipped segments solution:
  - performs efficiently
  - requires sender only deployment
- Future work: DETER?

Conclusion

- OptAck attack is dangerous: x1683
  - x32×10^6 w/ wscaling
- Deployed TCP Stacks are vulnerable
- Skipped segments solution:
  - performs efficiently
  - requires sender only deployment
- Future work: DETER?
- Questions?

This talk was written in $\LaTeX$ with Prosper: http://prosper.sourceforge.net/

Defense Categories

- bandwidth caps
  - just use more victims or more attackers
- nonces
  - non-cumulative nonces are not robust
  - cumulative nonces require client deployment
- ACK alignment/timestamps
  - requires CPU for strong random numbers
  - timestamps are optional
  - not robust to lazy attack

More Defenses

- In network support
  - not currently deployed
- Random Pauses
  - inefficient
- segment reordering
  - allows false positives from network reordering, dropped ACKs
  - no performance penalty