Automated Attack Discovery in TCP Congestion Control Using a Model-Guided Approach

Cristina Nita-Rotaru

PhD work of: Samuel Jero, Hyojeong Lee and Endadul Hoque, Purdue University

MS Thesis: Anthony Peterson, Northeastern University

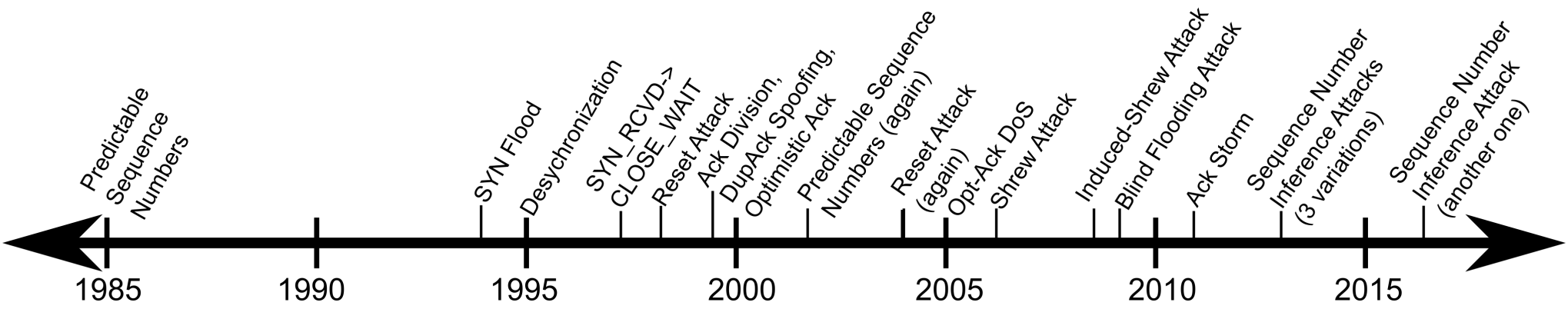
Khoury College of Computer Science

Northeastern University

TCP

- Transport protocol used by vast majority of Internet traffic
  - Including traffic encrypted with TLS
  - Including network infrastructure protocols like BGP
- Thousands of implementations
  - Over 5,000 implementation variants detectable by nmap
- Provides:
  - Reliability
  - In-order delivery
  - Flow control
  - Congestion control

TCP attacked for 30 years!



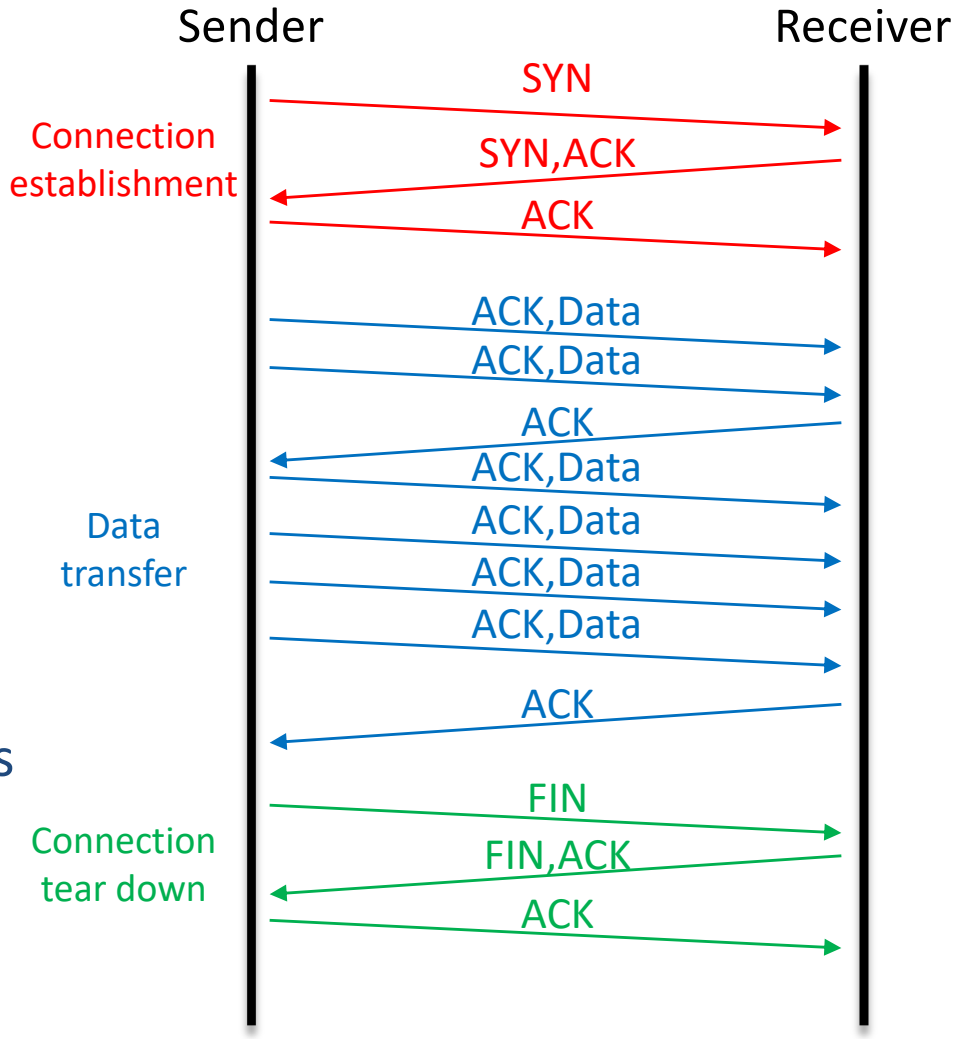
Why so many attacks?

- Complex goals
  - Reliability, in-order delivery, congestion control
- Many designs and implementations
  - Different designs for congestion control: Tahoe, Reno, New Reno, SACK, Vegas, BIC, CUBIC
  - Hundreds of implementations
- Written in low level languages
  - Highly efficient, but error-prone
- Heavily optimized
  - Prefers performance to ease of understanding and maintenance

Over 20 RFCs

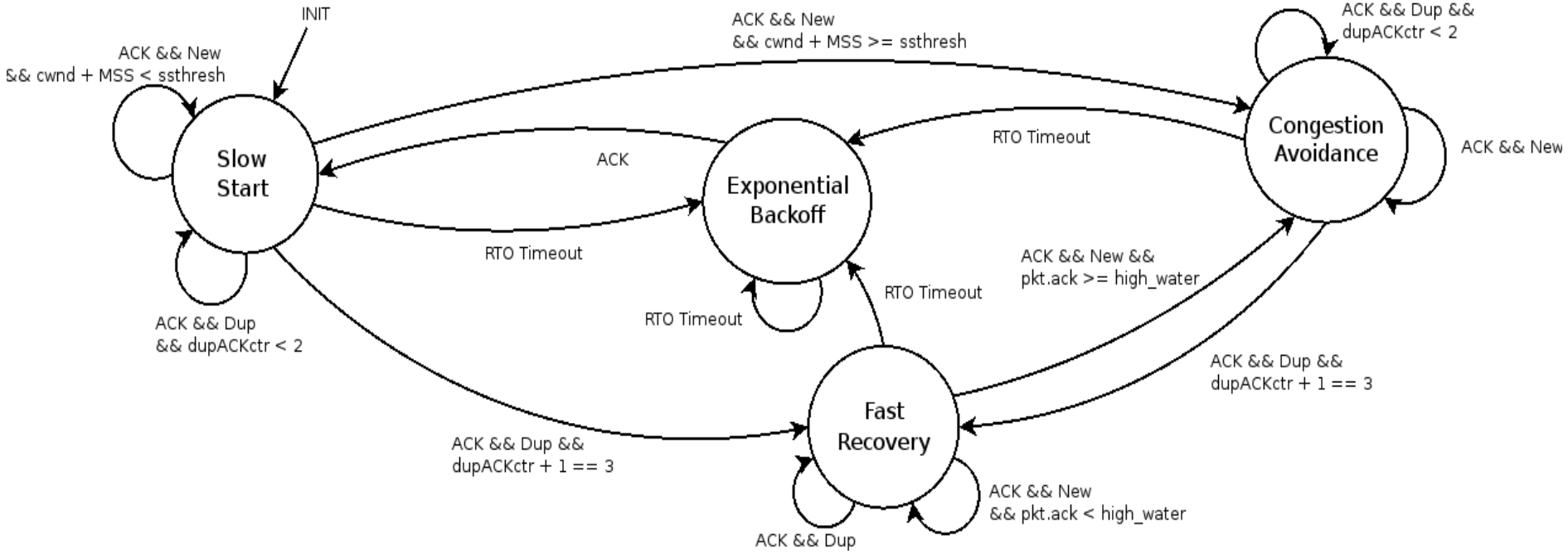
TCP connection set up and tear down

- Connection establishment
  - Check the other end exists
  - Set communication parameters on both directions
- Data sending
  - Bytes are numbered
  - Receiver periodically sends cumulative ACKS to sender
- Connection teardown
  - Graceful - each end releases its side of the connection
  - Abrupt - prevent attacks, lack on resources



TCP congestion control

- Protects against congestion collapse, provides fairness
- Many designs and implementations
  - Multiple Variations: Reno, New Reno, SACK, Vegas, BBR
  - Multiple Optimizations: PRR, TLP, DSACK, FRTO, RACK
  - Hundreds of implementations



This talk

Can we automatically find attacks in TCP implementations?
(without instrumenting the code)

- Connection establishment: SNAKE
  - State-machine based attack injection
- Congestion control: TCPwn (loss), aBBRate (model)
  - Model-based testing approach

Current methods

- Developer test suites
  - Tests used by developer to make sure implementation is cor
  - Packetdrill [USENIX 13]
- Fuzzi
  - KiF
  - Fin uts
- MAX [SIGCOMM 11]
  - Au
    exe

Ad-hoc, focused on benign scenarios

Difficulty reaching deep states, focus on crashes

Requires the user to select vulnerable lines of code

Attack model

Availability Attacks

- Keep resources allocated -- Denial of Service (DoS)
- Make a network service unavailable to all users
- Targeting a single connection using very focused actions

Performance Attacks

- Decrease the throughput of a target connection
- Stall a connection
- Increasing the throughout of a connection (basically making TCP behave as UDP – denial of service)

Our approach

- Test unmodified binaries in their native environment for close to deployment environment
- Testing for performance and availabilty issues, we need ts
- No code instrumentation
- Minimal input from user

1.Virtualization

2.Network emulation

5. AUTOMATED

3.Messages interception

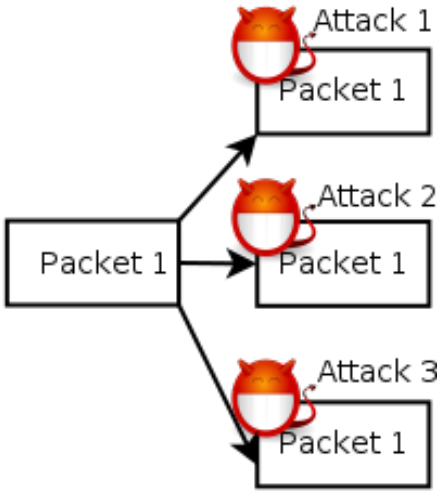
4.Message format, metric, topology, malicious nodes

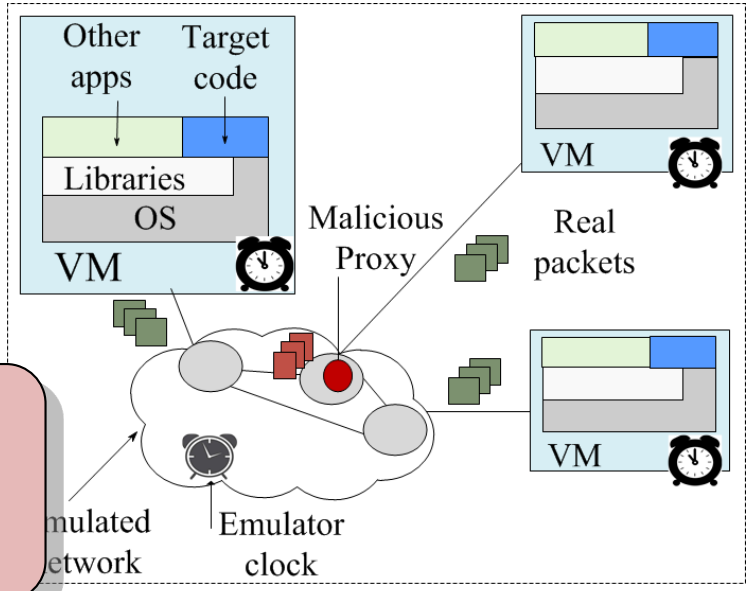
Design questions

- What attacks to create:
  - Disrupt message delivery: Delay, Divert, Duplicate, Drop:
  - Corrupt message content: lie field by field (based on field type range and on original value): Min and max, Zero, Scaling, Spanning, Random
- How to decide that the result was an attack:
  - Throughput, latency
- How to find attacks:
  - Brute force, greedy search algorithm, weighted greedy
- **When to inject an attack:**
  - Packet send-based, time-based, state machine-based

Attack injection: Packet send-based

- How
  - For each packet, inject each attack at packet send call
- Pros.
  - Simple
  - Systematic
- Cons.
  - Does not support injecting new packets

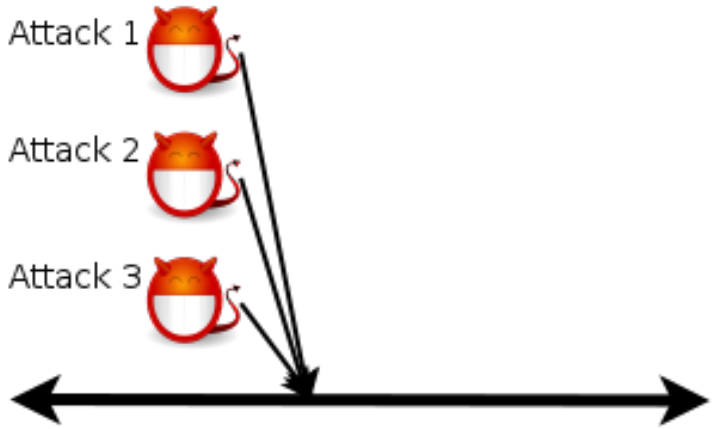




Not a good fit for finding attacks in TCP handshake

Attack injection: Time-based

- How
  - Every n seconds, inject a message attack and observe the result
  - Supports injecting new packets



Cannot achieve scalability and coverage!

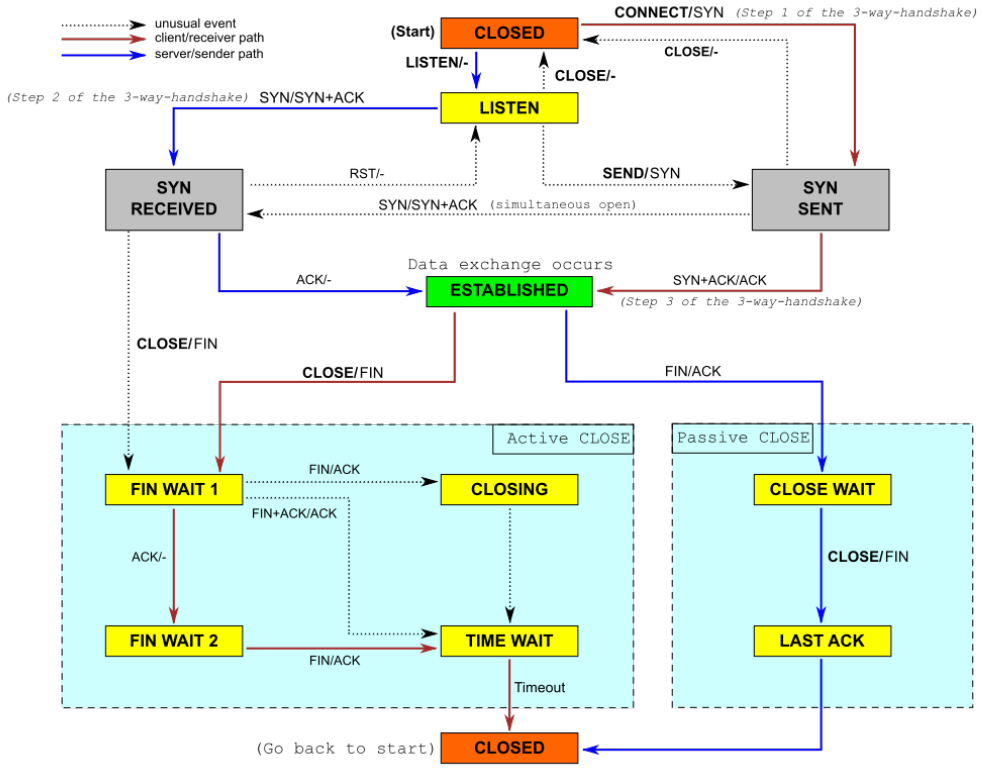
Scales with n*connection_length*attacks

- A minimum sized TCP packet takes 5 microseconds to transmit at 100Mbits/sec

12 million pkts*60 attacks*2min = 24 million hours

Our approach: Leverage state machine

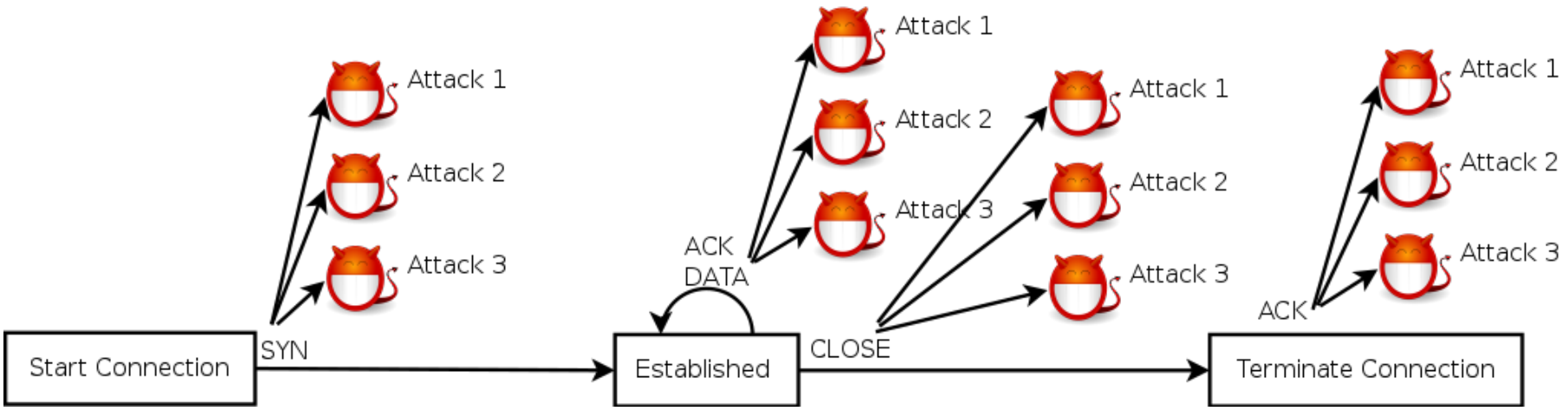
- Improved scalability and coverage
- State machine identifies key protocol areas
- Similar packet types received in the same state often perform similar actions
- Combine protocol state and packet type for attack injection



TCP Connection State Machine

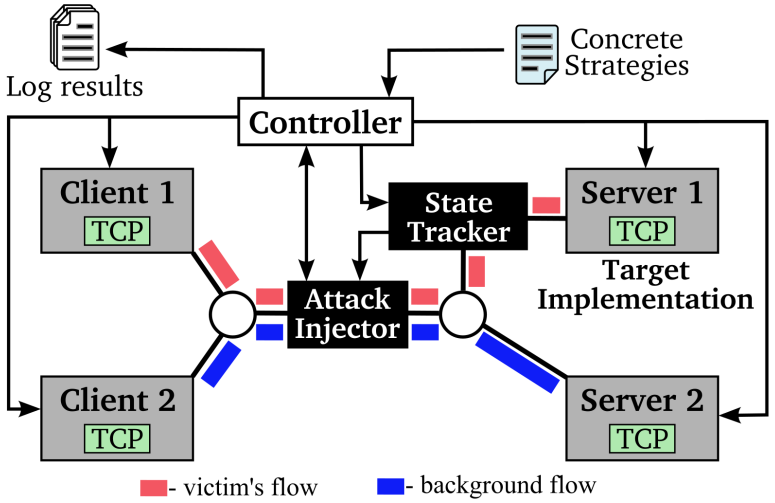
Attack injection: State machine-based

- How
  - Consider the protocol state, packet type pairs and apply each message attack to each pair
- Pros.
  - Scalable
  - Can apply attacks to more than a single packet
- Cons.
  - Assumes state machine is available
  - Assumes state machine is implemented correctly

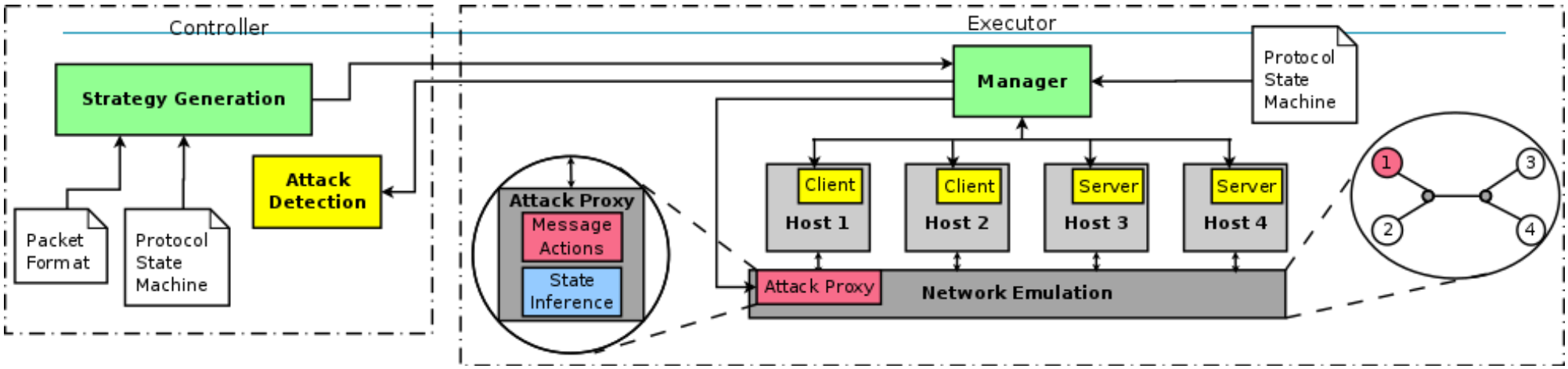


Decide if it was an attack or not

- During testing, performance and resource usage information collected to identify attacks
- Attack declared if:
  - Throughput of a flow is different that of the competing flow's by more than a factor of 2
  - Server resources are not released at the end of the test



SNAKE



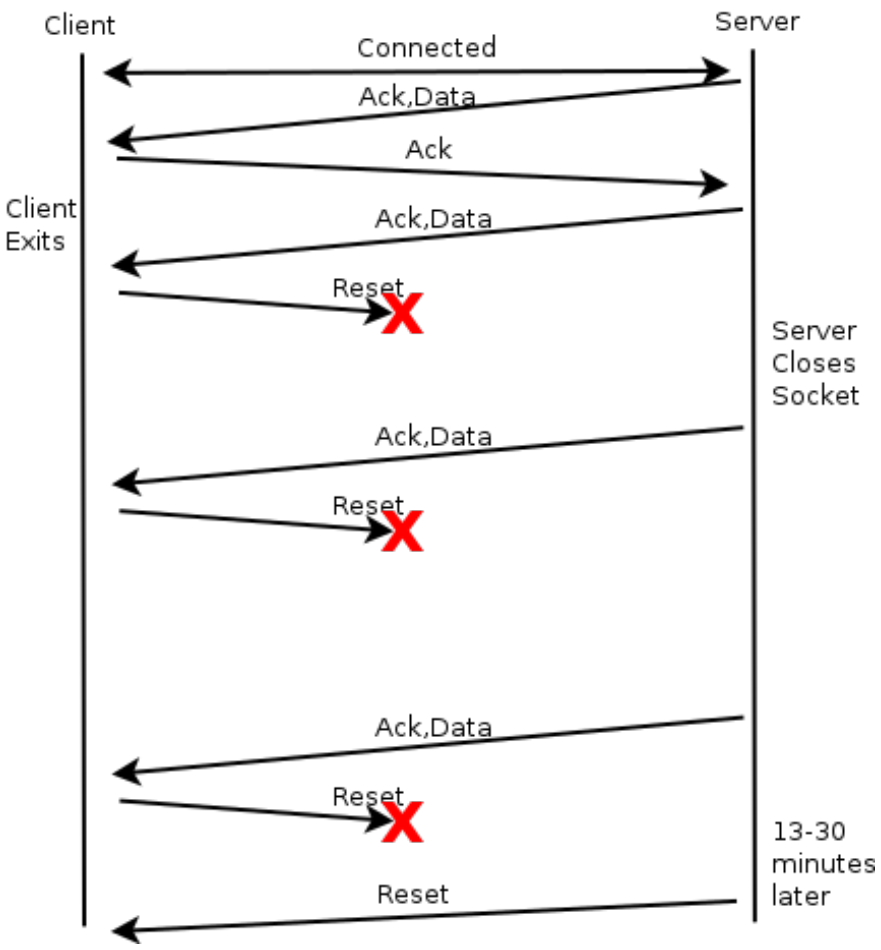
- Supported attacks: Drop, Duplicate, Delay, Batch, Reflect, Lie about packet fields, Inject, and HitSeqWindow
- Current protocol state tracked by monitoring packets

Leveraging State Information for Automated Attack Discovery in Transport Protocol Implementations
Samuel Jero, Hyojeong Lee, and Cristina Nita-Rotaru. DSN 2015. Best Paper Award.

TCP CLOSE_WAIT resource exhaustion attack

Client can force the server to keep socket state around for 13-30 minutes

- Client application exits
- Client responds to all future data with Resets
- Resets are dropped
- Server must receive ACKs for all data before it can close connection



TCP and DCCP

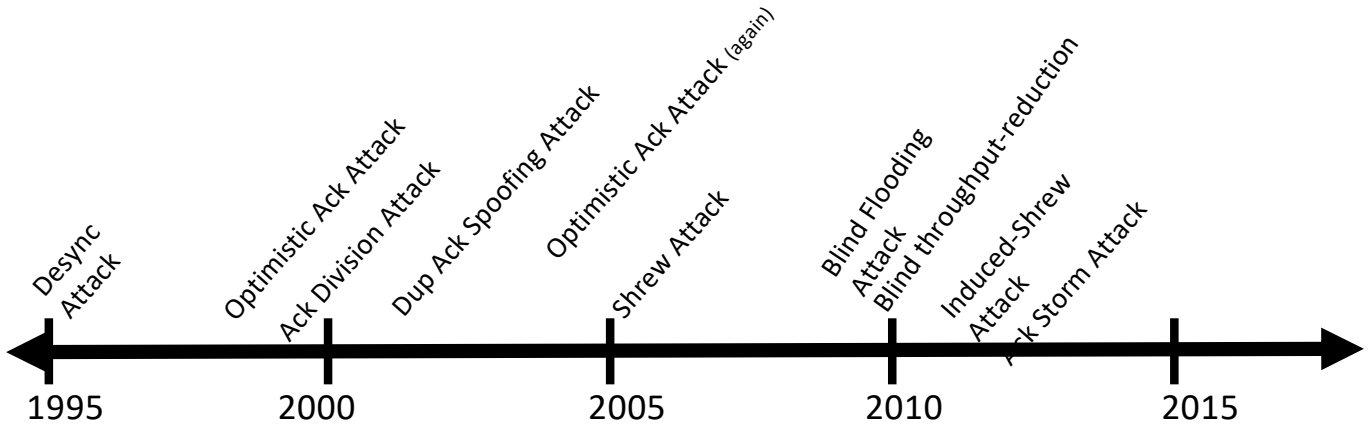
Protocol	Attack	Impact	OS	Known
TCP	CLOSE_WAIT Resource Exhaustion	Server DoS	Linux 3.0/3.13	Partially
TCP	Packets with Invalid Flags	Fingerprinting	Linux 3.0 / Win 8.1	No
TCP	Duplicate Ack Spoofing	Poor Fairness	Win 95	Yes
TCP	Reset Attack	Client DoS	All	Yes
TCP	SYN-Reset Attack	Client DoS	All	Yes
TCP	Duplicate Ack Rate Limiting	Degraded Throughput	Win 8.1	No
DCCP	Ack Mung Resource Exhaustion	Server DoS	Linux 3.13	No
DCCP	In-window Ack Sequence Number Modification	Degraded Throughput	Linux 3.13	No
DCCP	REQUEST Connection Termination	Client DoS	Linux 3.13	No

This talk

Can we automatically find attacks in TCP implementations?
(without instrumenting the code)

- Connection establishment: SNAKE
  - State-machine based attack injection
- Congestion control: TCPwn (loss), aBBRate (model)
  - Model-based testing approach

Congestion control-related attacks



Attacks may result in:

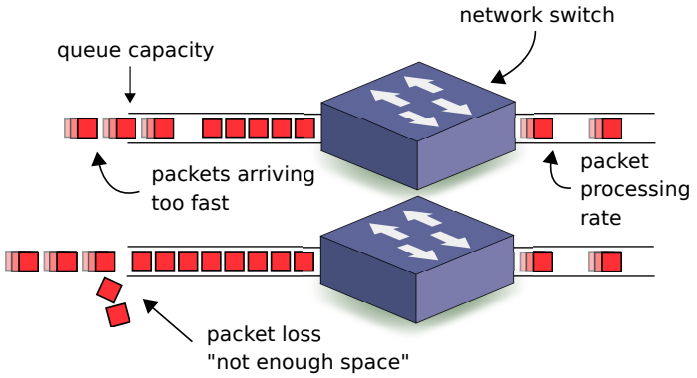
- **Decreased** throughput
- **Increased** throughput that starves competing flows
- **Stalled** data transfer

SNAKE could not find these attacks

Congestion control

- Goals
  - Avoid overwhelming the network
  - Divide bandwidth to flows sharing network
- How
  - Use signals from network to detect congestion
- TCP Congestion Control
  - Leverages acknowledgments (ACKs) to detect packet loss

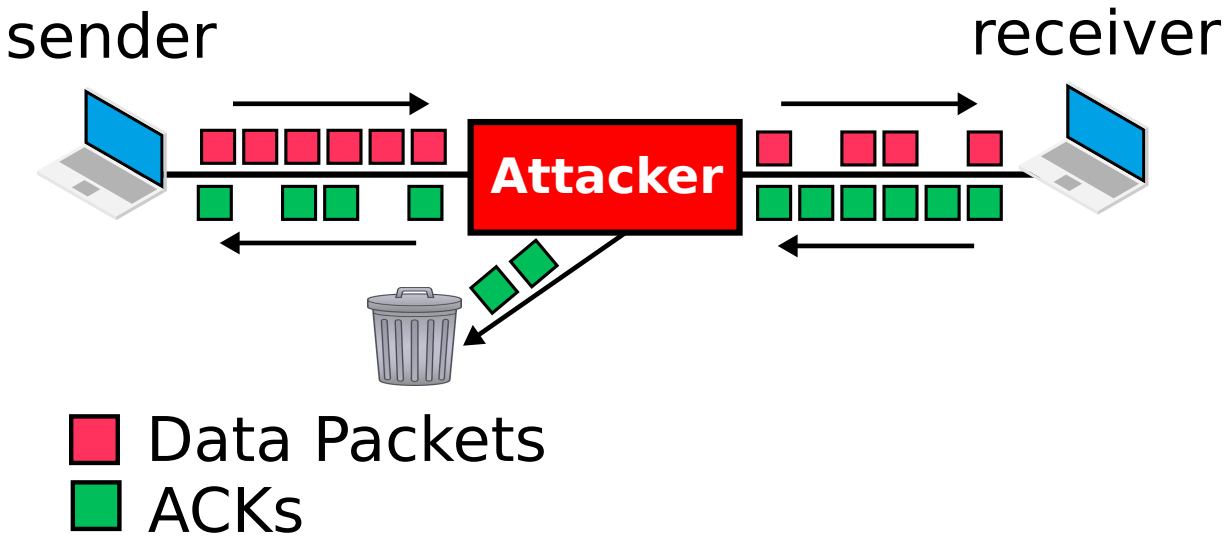
Packet loss as congestion signal



How it works: Some queue along path overflowed due to congestion

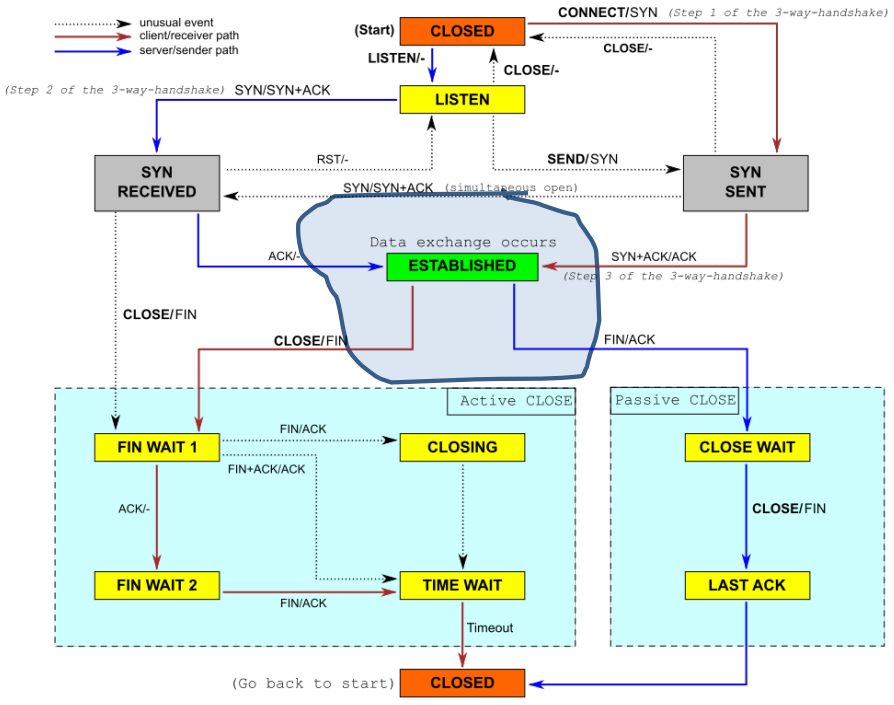
ACK manipulation attacks

- Manipulate ACKs to fool sender about actual congestion
- Attacker needs to observe (on-path) or predict (off-path) a sequence to be able to inject packets
- Cause sender to **send too fast, too slow or stall**



Why SNAKE could not find those attacks?

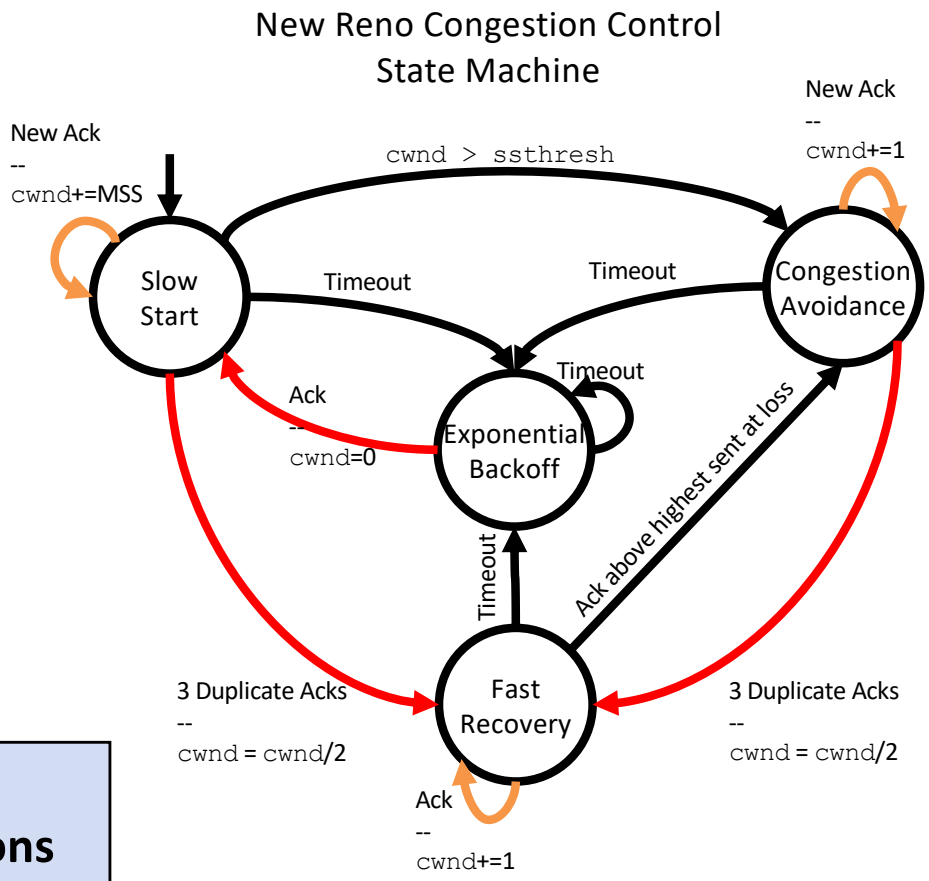
- State machine we modeled perceived congestion control as a black-box, we modeled only connection establishment, steady state was modeled as one state
  - No visibility into ACKs and their relation to the different stages of congestion control



Optimistic Ack Attack

How does it work: Increase sending rate by acknowledging data that has not been received yet

- Acknowledging new data causes yellow transitions to be taken
- Increases `cwnd` and thus throughput with each loop
- Avoids red transitions which reduce `cwnd` and thus throughput

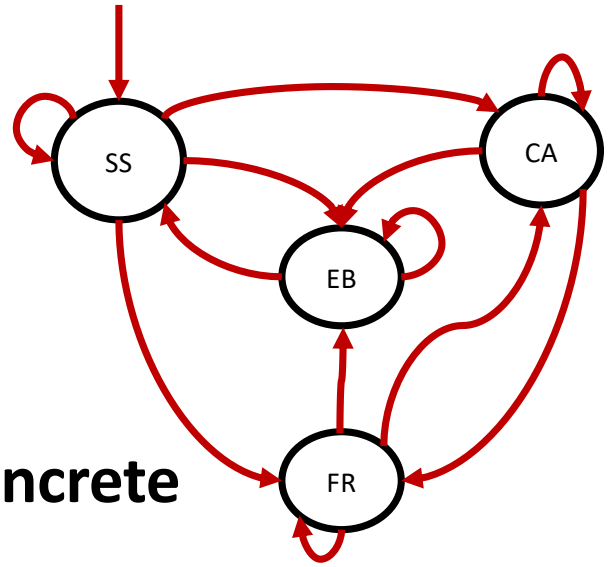


Key Takeaways:

- **Attacks attempt to cause desirable transitions**
- **Attacks must repeatedly execute transition to have noticeable impact**

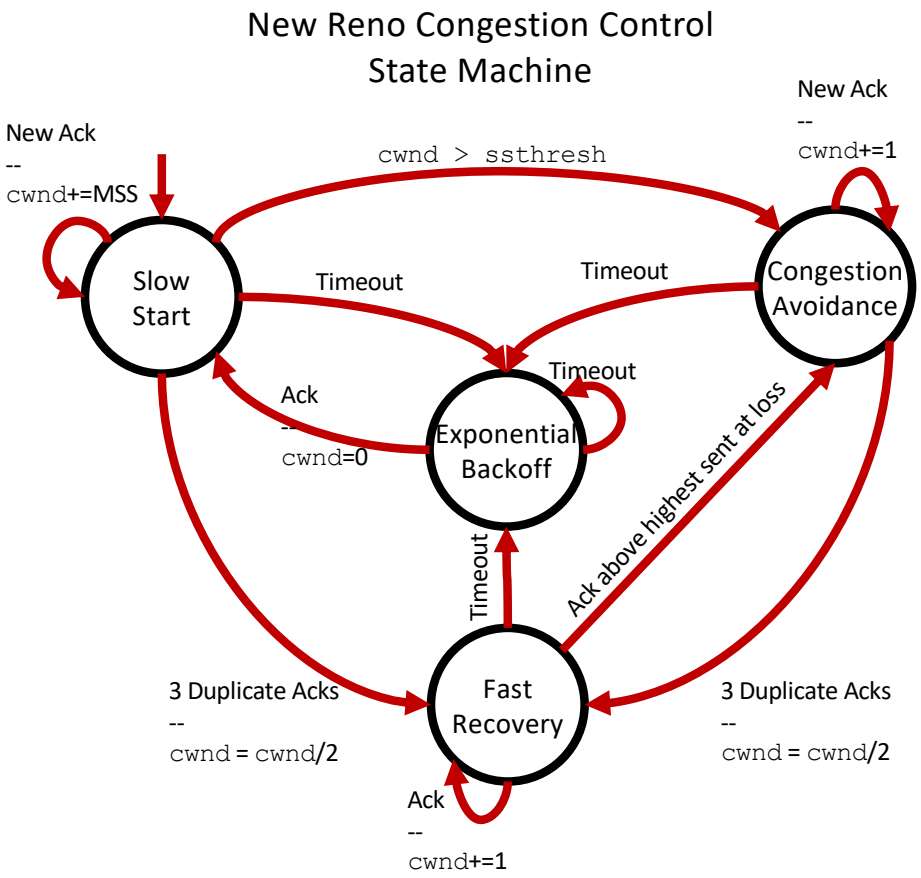
Our approach for congestion control: TCPwn

- Use **model-based testing** to identify all possible attacks in a *scalable* manner
  - Use an abstract model to generate abstract strategies
  - Map abstract strategies to concrete strategies
  - Execute concrete strategies on implementations to find attacks causing:
    - Decreased throughput
    - Increased throughput
    - Connection stall
- **1. How to select the abstract model**
- **2. How to find abstract strategies**
- **3. How to map abstract strategies to concrete strategies**



Our model: New Reno

- State machine
  - Input: Acks and Timers
  - Output: Congestion Window (`cwnd`) = sending rate
- Four states:
  - *Slow Start*—Quickly find available bandwidth
  - *Congestion Avoidance*—Steady state sending with occasional probe for more bandwidth
  - *Fast Recovery*—React to loss by slowing down
  - *Exponential Backoff*—Timeout, slow down



Why New Reno

- General-enough state machine
- It is the starting based for _most_ TCP congestion control algorithms
- Does not capture optimizations, but our results show that was good enough in practice
- (As we will show later we need to also be able to infer the state at ran time so simpler is better)
- We trade-off precision for generality

Model-based attack generation

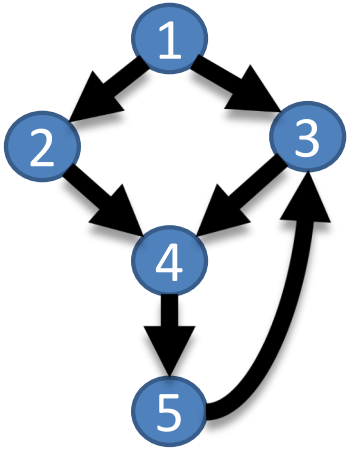
Generate all cycles with the following pattern:

- **`cwnd` increases/decreases along cycle**
- **A set of actions exist that force TCP to follow this cycle**

1. Consider state machine model of congestion control
2. Identify cycles containing desirable transitions
   - Abstract strategy generation
3. Force TCP to follow each cycle
   - Concrete strategy generation

Abstract strategy generation

- Enumerate all paths
  - No standard graph algorithm
  - We adapt *depth first search* to this problem
- Check that path contains cycle
- Check that cycle contains desirable transitions
  - Any change to `cwnd`
- Add path and transition conditions to abstract strategies



✓ **Cycle**

✓ **Desirable Transition**

Abstract strategies are merely desirable cycles; they may not be realizable in practice!

From abstract to concrete strategies

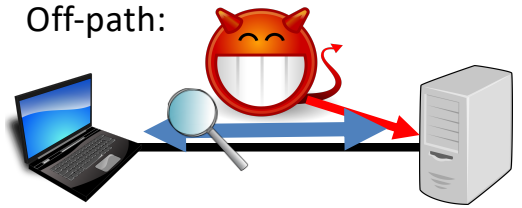
We want to test implementations

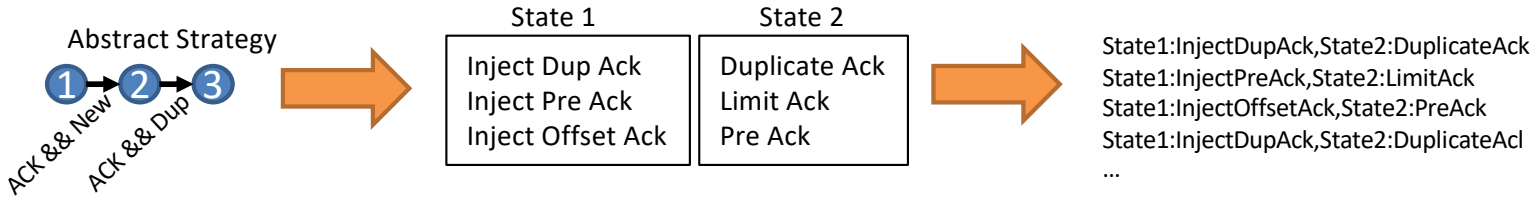
- Limited to packet manipulation and injection to cause abstract strategies
- Consider each abstract strategy separately
- Map each transition to a set of basic malicious actions
  - Actions chosen to cause transition
  - Based on attacker capabilities

Attacker Types:

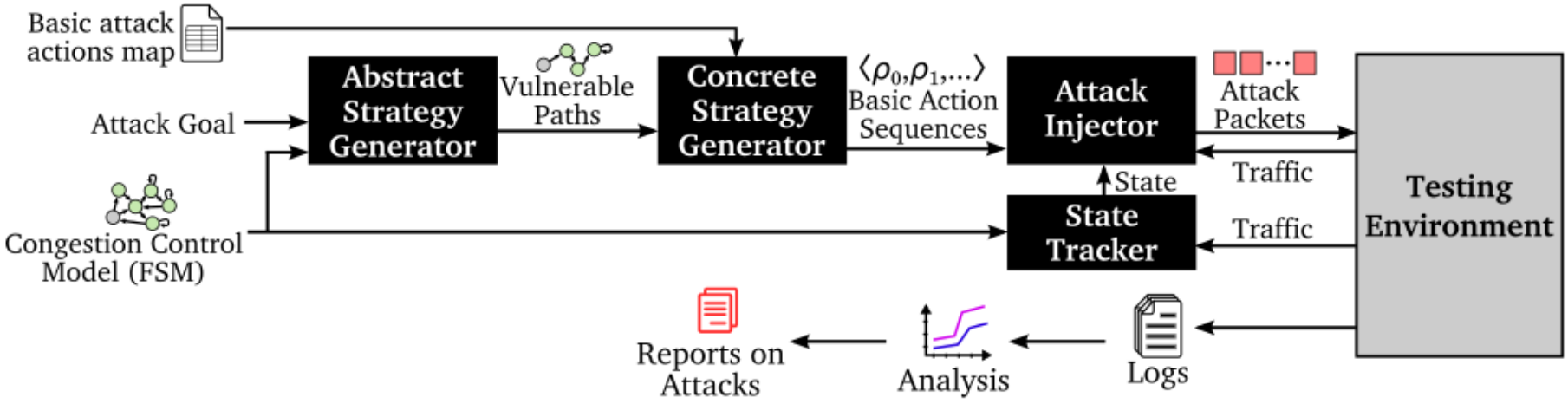
On-path:

Off-path:

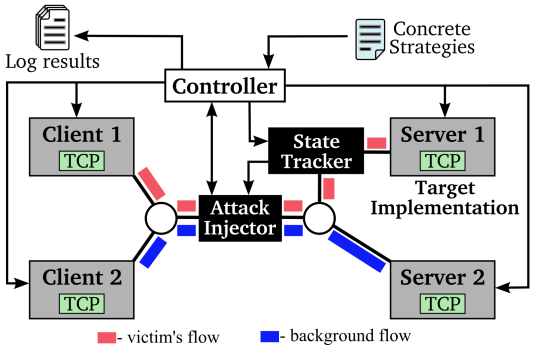




TCPwn design



- Test strategies created using model-based testing and our abstract and concrete strategy generators
- Attack injector applies malicious actions
- Performance of target TCP connection identifies attacks



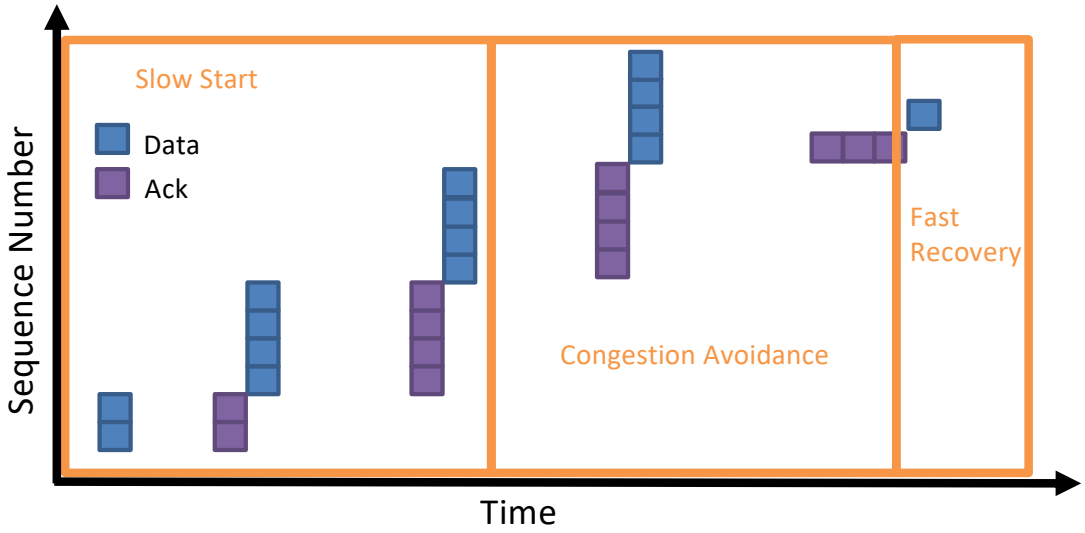
Automated Attack Discovery in TCP Congestion Control Using a Model-guided Approach. S. Jero, E. Hoque, D. Choffnes, A. Mislove, C. Nita-Rotaru. NDSS 2018, CISCO Network Security Distinguished Paper Award

Inferring congestion control state

To apply concrete strategies to an implementation, we need to know the sender's congestion control state

- Approximate congestion control state and assume normal application behavior
- Take a small timeslice and observe the bytes sent and acknowledged by the implementation

Bytes Sent*2 ≈ Bytes Acked
State: Slow Start
Bytes Sent ≈ Bytes Acked
State: Congestion Avoidance
Retransmitted packets or ACK pkts > Data pkts
State: Fast Recovery
ACK pkts == 0 and Data pkts > 0
State: Exponential Backoff



Evaluation

We tested five TCP implementations:

Implementation	Date	Congestion Control
Ubuntu 16.10 (Linux 4.8)	2016	CUBIC+SACK+FRTO+ER+PRR+TLP
Ubuntu 14.04 (Linux 3.13)	2014	CUBIC+SACK+FRTO+ER+PRR+TLP
Ubuntu 11.10 (Linux 3.0)	2011	CUBIC+SACK+FRTO
Debian 2 (Linux 2.0)	1998	New Reno
Windows 8.1	2014	Compound TCP + SACK

Found 11 classes of attacks, 8 of them unknown

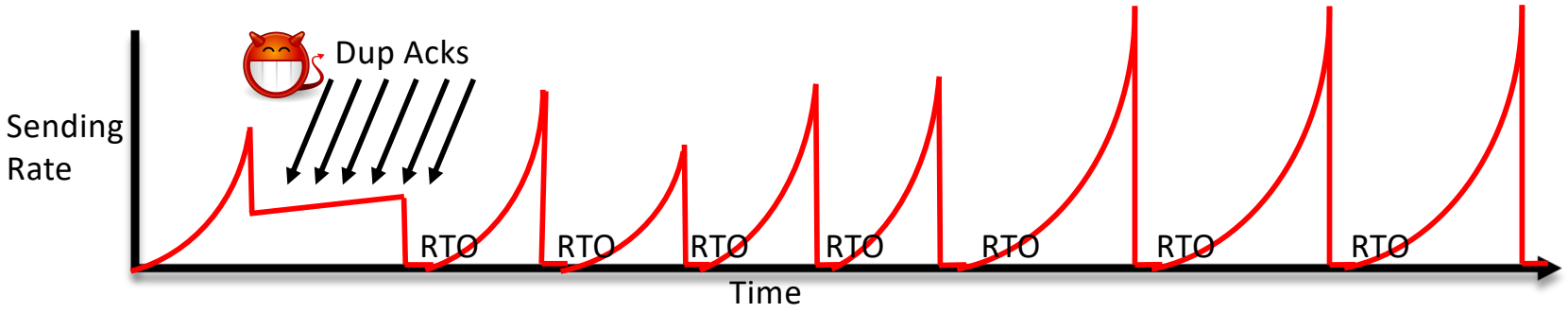
Results summary

Attack Class	Attacker	Impact	OS	New?
Optimistic Ack	On-path	Increased Throughput	ALL	No
On-path Repeated Slow Start	On-path	Increased Throughput	Ubuntu 11.10, Ubuntu 16.10	Yes
Amplified Bursts	On-path	Increased Throughput	Ubuntu 11.10	Yes
Desync Attack	Off-path	Connection Stall	ALL	No
Ack Storm Attack	Off-path	Connection Stall	Debian 2, Windows 8.1	No
Ack Lost Data	Off-path	Connection Stall	ALL	Yes
Slow Injected Acks	Off-path	Decreased Throughput	Ubuntu 11.10	Yes
Sawtooth Ack	Off-path	Decreased Throughput	Ubuntu 11.10, Ubuntu 14.04, Ubuntu 16.10, Windows 8.1	Yes
Dup Ack Injection	Off-path	Decreased Throughput	Debian 2, Windows 8.1	Yes
Ack Amplification	Off-path	Increased Throughput	Ubuntu 11.10, Ubuntu 14.04, Ubuntu 16.10, Windows 8.1	Yes
Off-path Repeated Slow Start	Off-path	Increased Throughput	Ubuntu 11.10	Yes

Off-path repeated slow start attack

- Linux includes adjustable dup ack threshold
  - Based on observed duplicate and reordered packets
- Attacker injects many duplicate acks
  - Increasing dup ack threshold
- Timeout occurs before dup ack loss detection
- Enter Exponential Backoff and then Slow Start
  - Instead of Fast Recovery
- Short 200ms timeout causes throughput to be >= normal
- Competing connections also suffer badly due to repeated losses

Off-path attacker can increase throughput for Linux senders



Discussion

- Use of New Reno as model
  - Model limited by ability to infer sender's state from network traffic
  - More precise inference or instrumentation would enable more precise modeling
  - We trade off precision for ease of application to a wide range of implementations
- What about CUBIC, SACK, etc?
  - Most algorithms/optimizations are similar to New Reno (includes: SACK, CUBIC, TLP, PRR)
  - We actually tested implementations of these and found attacks
- What about algorithms not similar to New Reno?
  - For example: BBR, TFRC, Vegas
  - Model-based testing still readily generates abstract strategies
  - Need a method to infer sender's congestion control state

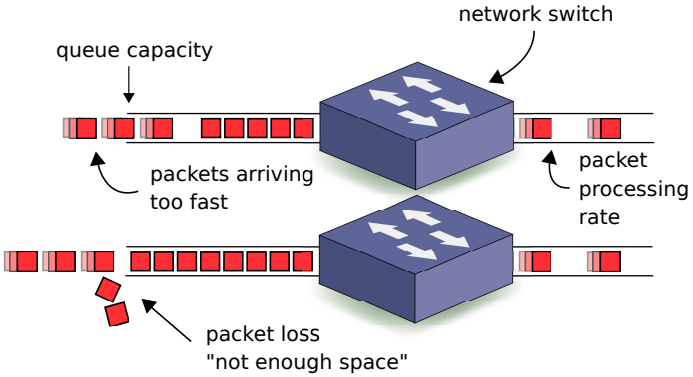
This talk

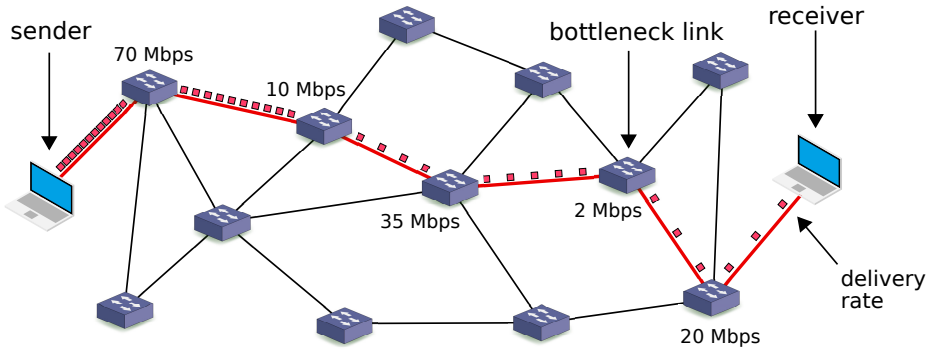
Can we automatically find attacks in TCP implementations?
(without instrumenting the code)

- Connection establishment: SNAKE
  - State-machine based attack injection
- Congestion control: TCPwn (loss), aBBRate (model)
  - Despite using a different congestion control approach, is BBR prone to acknowledgment-based manipulation attacks?
  - Are there any known attacks that BBR is immune to?

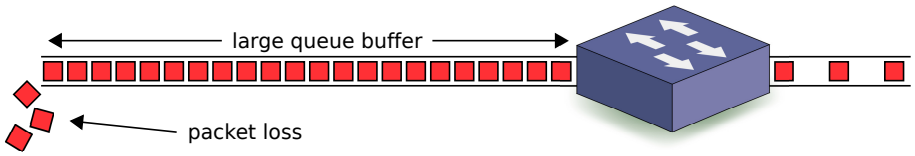
BBR: Motivation

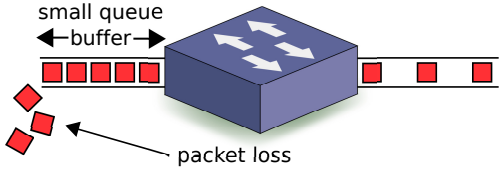
Packet loss as congestion signal





How it works: Some queue along path overflowed due to congestion





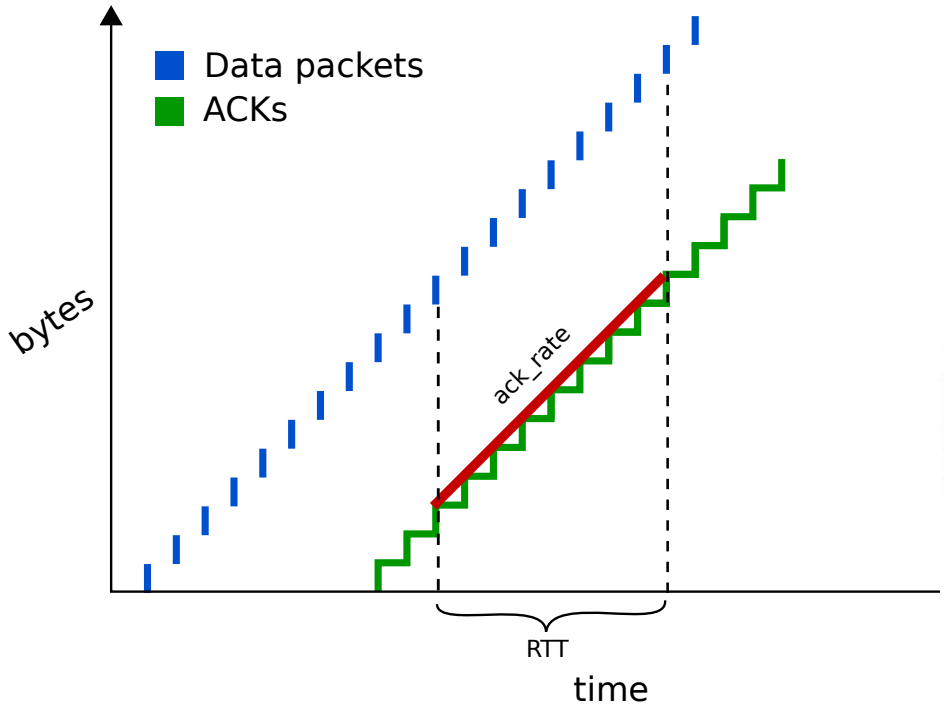
What is actually needed is to estimate the bottleneck link and not send faster than that

In modern networks: less effective

BRR: Congestion Control

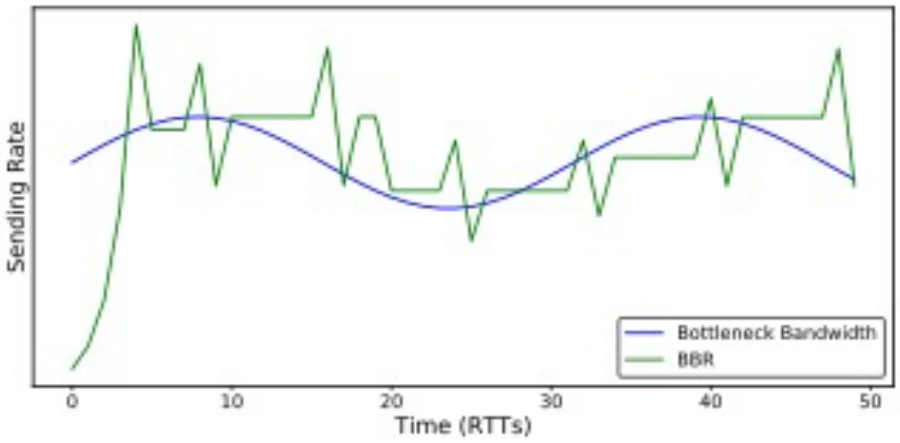
Detecting congestion

- Estimates the Bottleneck Bandwidth
- By measuring fine-grained ACK rate across RTT intervals



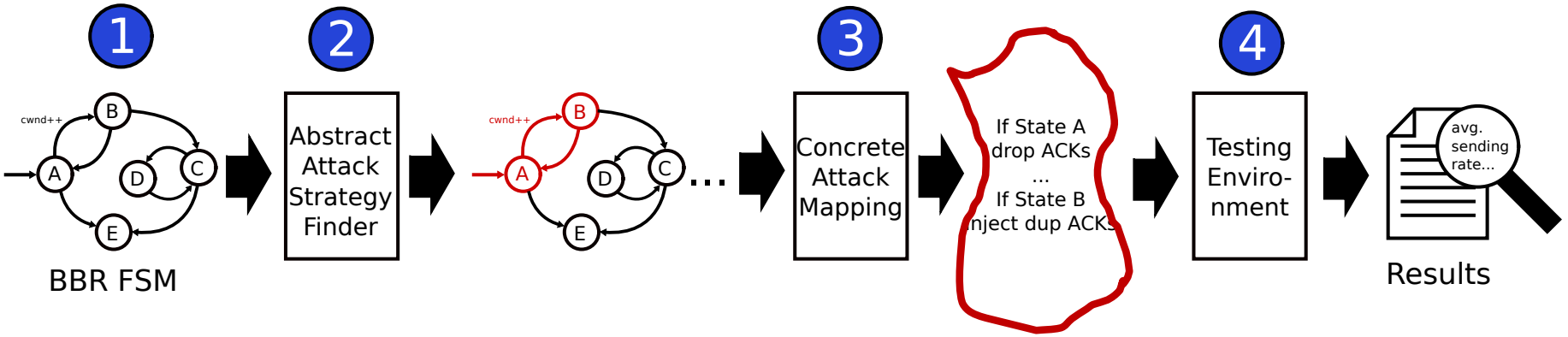
Reacting to congestion:

- Retain max delivery rate sample for 10 RTTs, and send proportionally
- Send 25% faster 1/8 RTTs to approach network limit
- Backs off from network when old max delivery rate sample expires



Adapting TCPwn for BBR

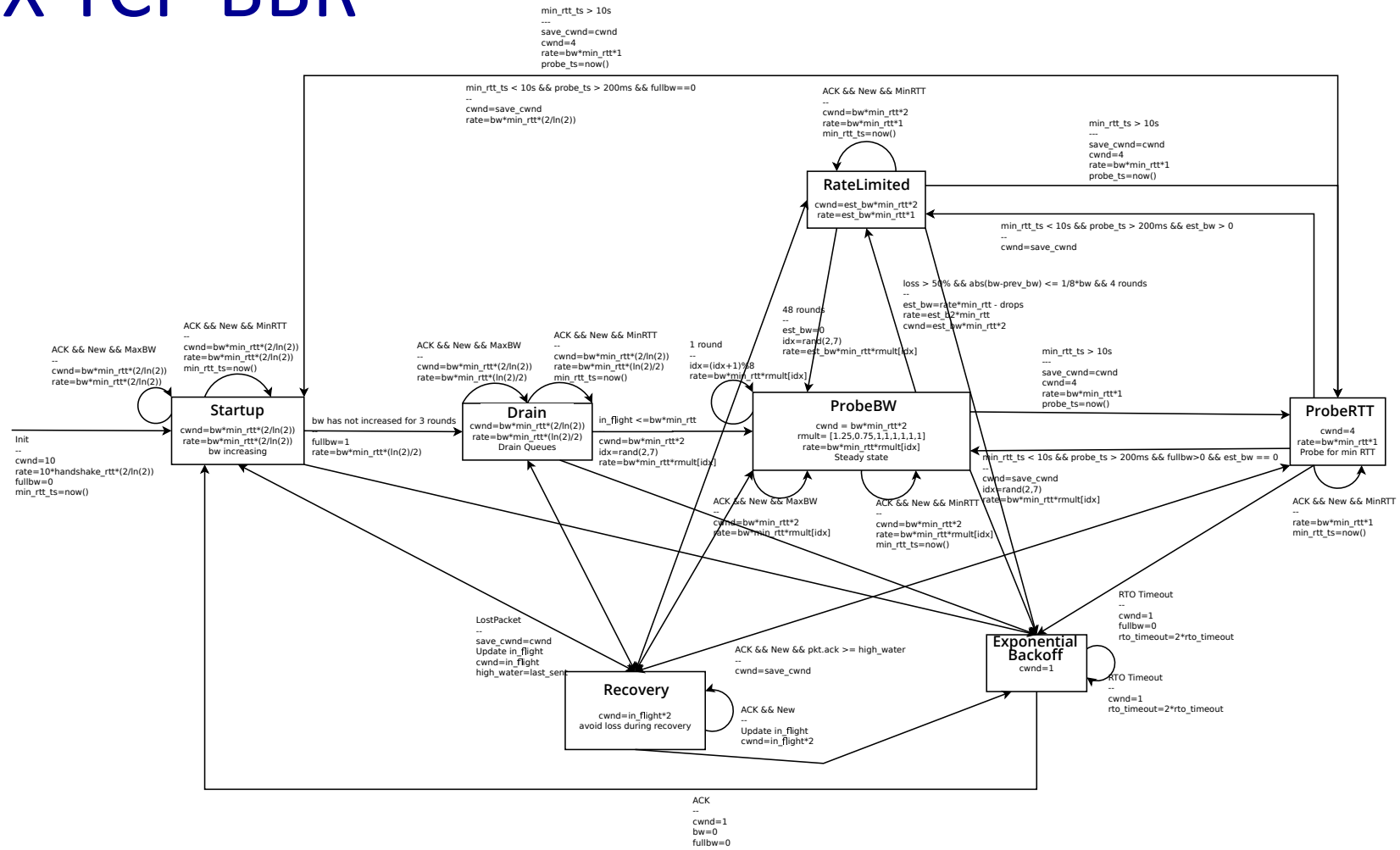
- TCPwn use the congestion control finite-state machine (FSM) encoding
- Searches for all paths in FSM that manipulate its sending rate
- Map paths to actual attacks
- Execute attack while measuring sending rate



On-path concrete attacks supported

- ACK duplication: send same ACK several times
- ACK stepping: several ACKs are dropped and then several let through in a cycle
- ACK bursting: ACKss are sent in bursts
- Optimistic ACK: acknowledge highest byte, dropping duplicates
- Delayed ACK: delay ACKs for a fixed amount of time
- Limited ACK: prevent ACK numbers from increasing
- Stretch ACK: forward only every n^{th} ACK
- Off-path: ACK duplication, offset acknowledgments, and incrementing ACKS

LINUX TCP BBR



- We extracted BBR FSM from code (not available anywhere)

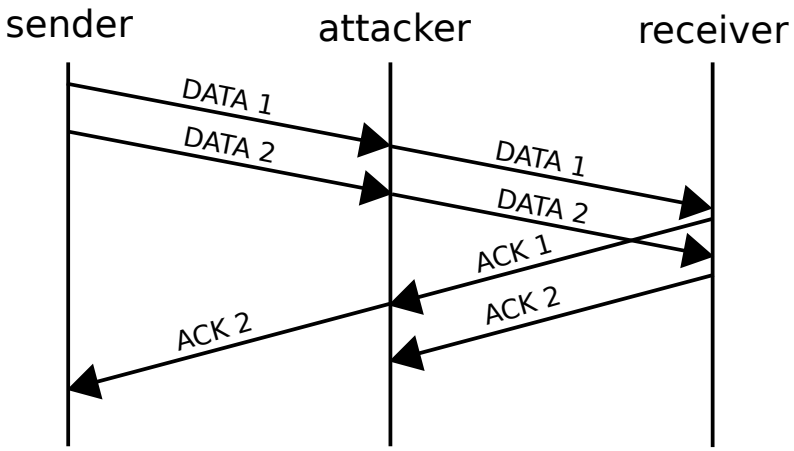
aBBRate: Automating BBR Attack Exploration Using a Model-Based Approach A. Peterson, S. Jero, E. Hoque, D. Choffnes, C. Nita-Rotaru. RAID 2020

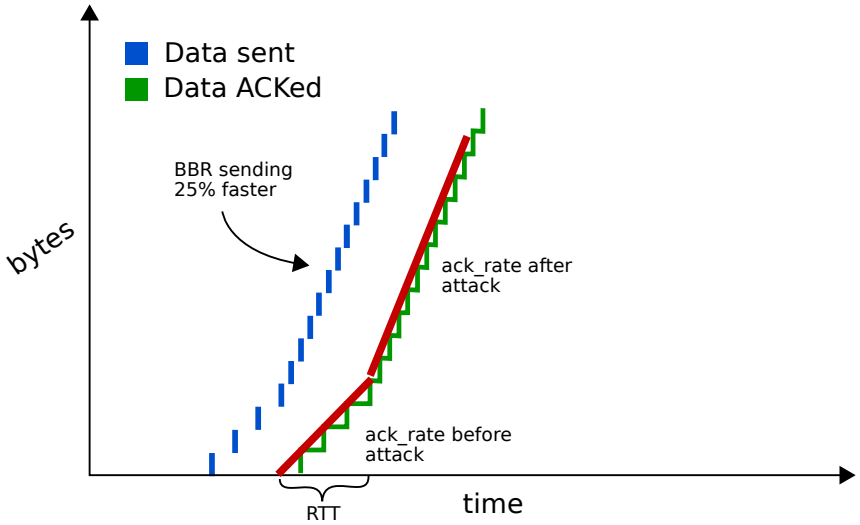
Evaluation

- Generated 30,297 attack strategies
- 8,859 caused faster, slower or stalled connections
- 14 – Faster
  4,025 – Slower
  4,820 – Stalled (transmission halts)
- 5 classes of attacks

Attack class	Impact
Optimistic acknowledgments	Faster
Delayed acknowledgments	Slower
Repeated Re-transmission timeout	Slower
Re-transmission timeout stall	Stalled
Sequence number de-sync stall	Stalled

Optimistic ACK

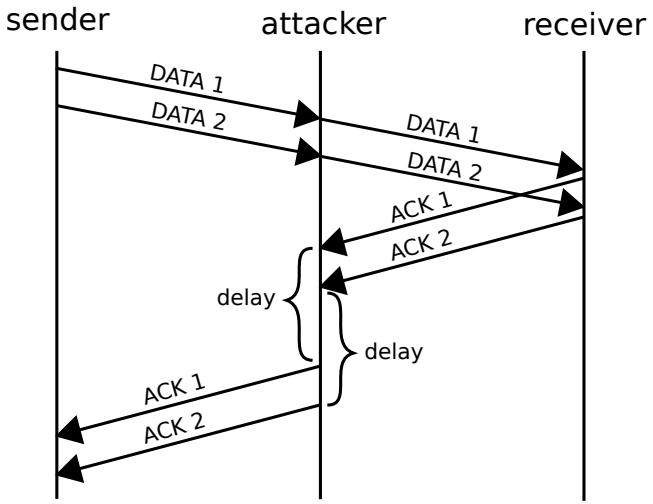


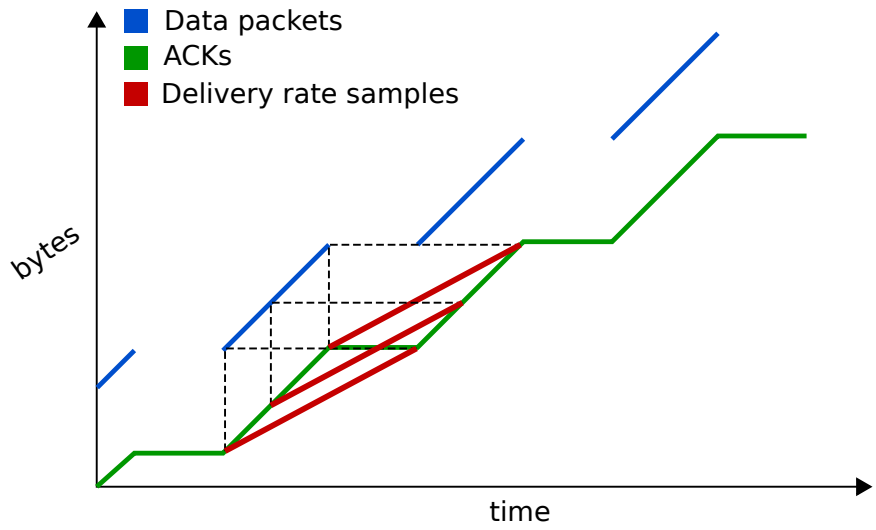


ACK highest observed data sequence, makes the sender send faster

- When BBR probes for bandwidth, it sends 25% faster for 1 RTT
- ACK rate follows the increased sending rate
- BBR believes the network can sustain the increased rate

Delayed ACK

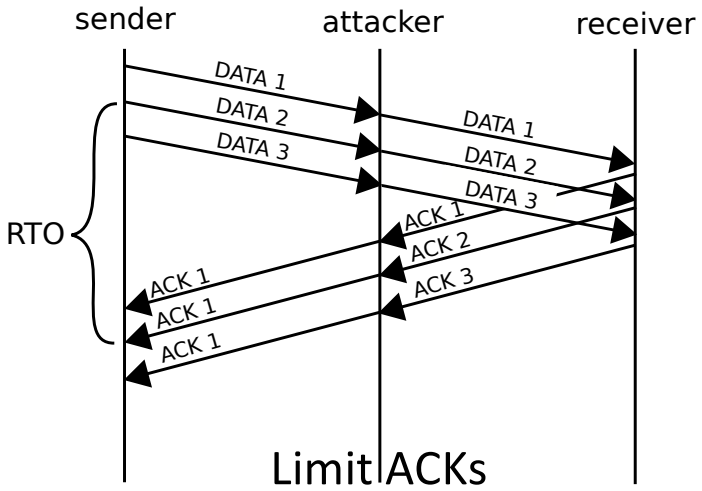


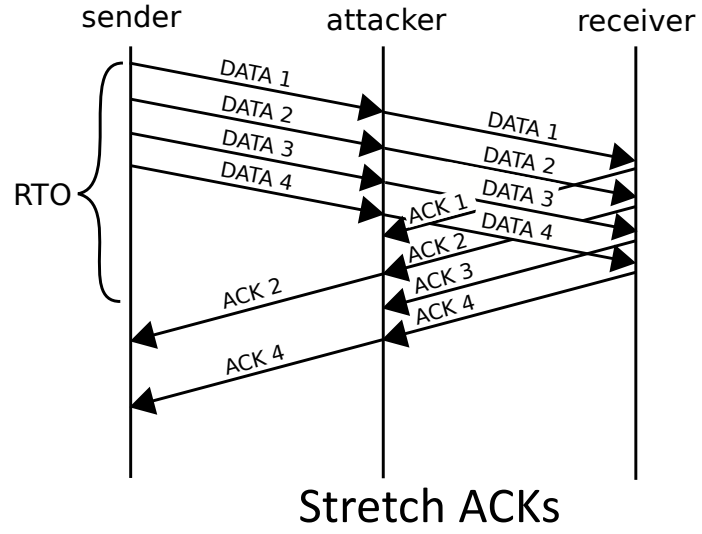


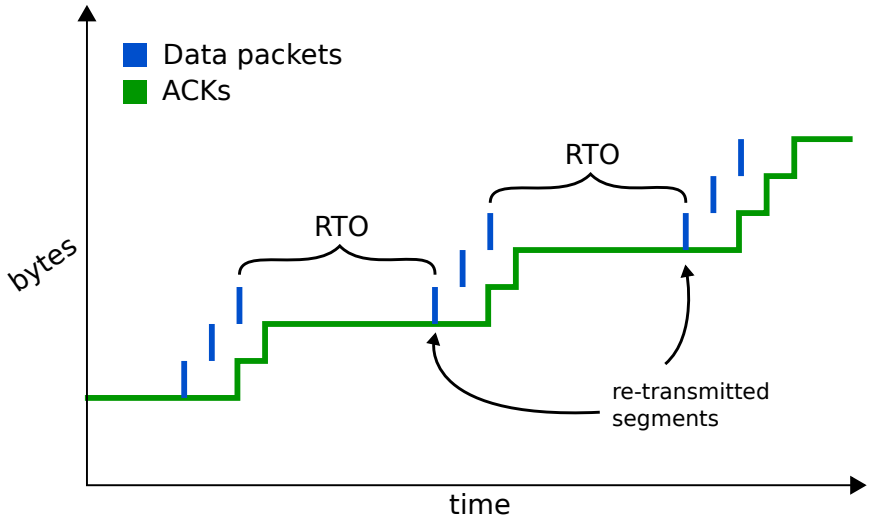
Delay ACKs for a constant amount of time

- Initial delay in ACKs at sender
- Sender stops sending data because no data is ACKed
- BBR believes the bottleneck bandwidth is smaller than reality
- Takes effect after 10 RTTs, due to bottleneck bandwidth filter

Repeated retranssmission timeout

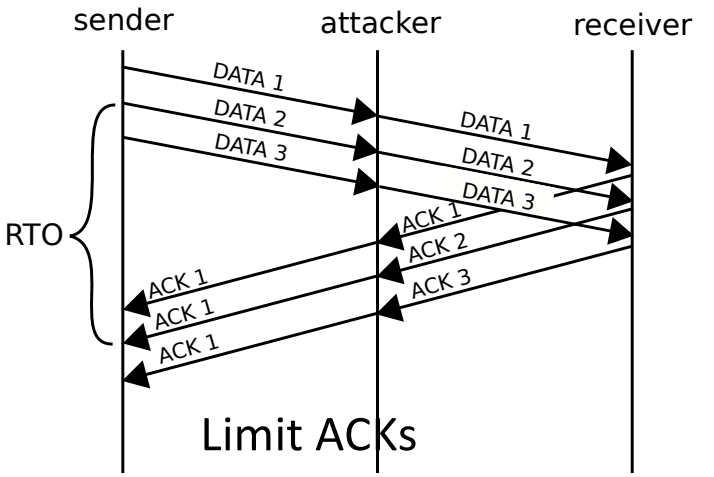


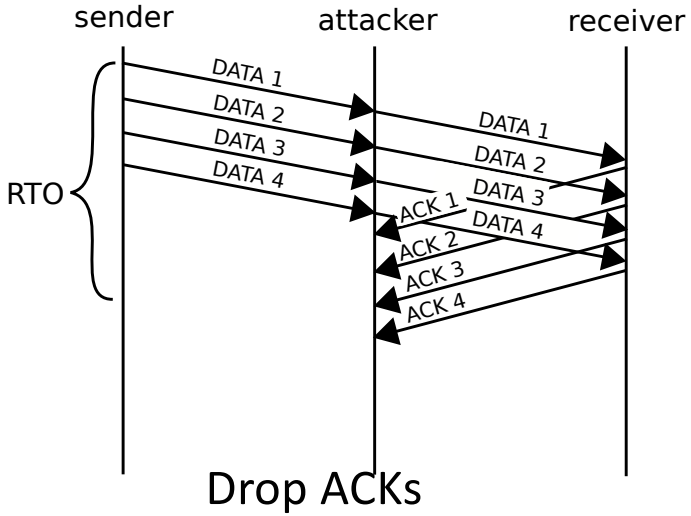


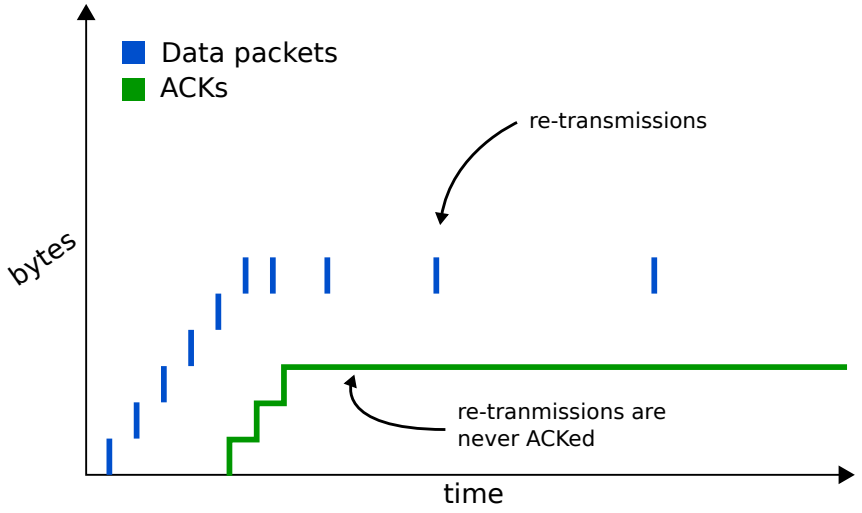


- Until RTO occurs, sender waits and does not send data
- A lot of time is wasted idling
- Data is sent in small bursts between RTOs

Re-transmission timeout Stall



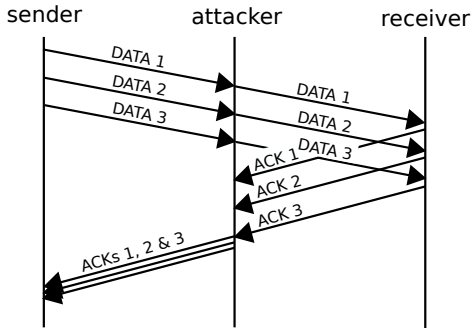




- If no new data is ACKed, no new data will be sent

Attacks ineffective against TCP BBR

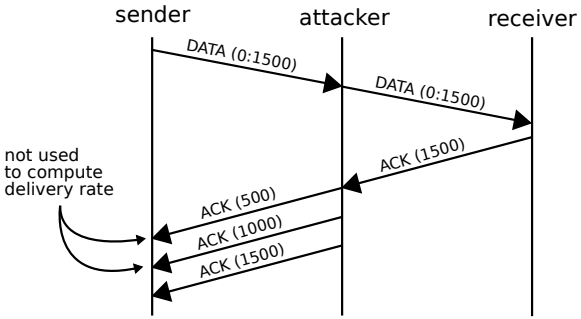
ACKs Bursts



Attacker accumulates n ACKs and sends in a burst

BBR immune because it sends data proportional to average ACK rate over RTT intervals

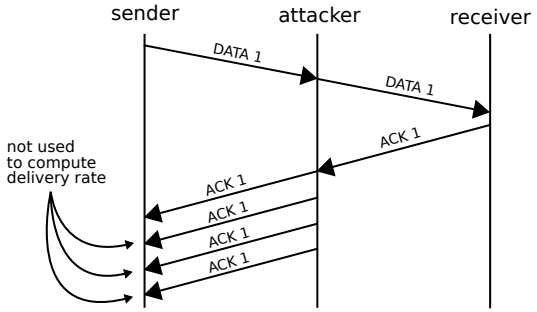
Divided ACKs



Attacker divides single ACK into n smaller ACKs

BBR immune because it only computes delivery rate samples for segments that the ACK specifically acknowledges

Duplicate ACKs



Duplicate single ACK n times

BBR immune because it does not use packet loss to signal congestion, duplicate ACKs are not used to compute delivery rate samples

Summary

- Show how to find automatically attacks in TCP implementations without instrumenting the code
  - Connection establishment
  - Congestion control

Check out the code!
https://github.com/samueljero/snake
https://github.com/samueljero/TCPwn

ISEC Building

Acknowledgments

- Partially sponsored by NSF SaTC 1421815-CNS, NSF SATC CNS-1223834
- **PhD thesis work of: Samuel Jero, MS Thesis Antony Peterson, contributions by Endadul Hoque, Hyojeong Lee.**

Relevant publications

- **Leveraging State Information for Automated Attack Discovery in Transport Protocol Implementations** Samuel Jero, Hyojeong Lee, and Cristina Nita-Rotaru. 45th IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), Jun. 2015. Best Paper Award.
- **Automated Attack Discovery in TCP Congestion Control Using a Model-guided Approach.** Samuel Jero, Endadul Hoque, David Choffnes, Alan Mislove, Cristina Nita-Rotaru. NDSS 2018, Feb. 2018. CISCO Network Security Distinguished Paper Award
- **aBBRate: Automating BBR Attack Exploration Using a Model-Based Approach** Anthony Peterson, Samuel Jero, Endadul Hoque, Dave Choffnes, Cristina Nita-Rotaru. RAID 2020