BFT: From Z To A

Zyzzyva

BFT: From Z To A

Aardvark

Making Byzantine
Fault Tolerant Systems
Tolerate Byzantine Faults

Paved with good intentions

- No BFT protocol should rely on synchrony for safety
- FLP: No consensus protocol can be both safe and live in an asynchronous system
  - All one can guarantee is eventual progress

Paved with good intentions

- No BFT protocol should rely on synchrony for safety
- FLP: No consensus protocol can be both safe and live in an asynchronous system
  - All one can guarantee is eventual progress
- "Handle normal and worst case separately as a rule, because the requirements for the two are quite different:
  the normal case must be fast;
  the worst case must make some progress"
  -- Butler Lampson, "Hints for Computer System Design"

The road more traveled

- Maximize performance when
  - the network is synchronous
  - all clients and servers behave correctly
- While remaining
  - safe if at most f servers fail
  - eventually live

The Byzantine Empire (565 AD)

Synchronous, no failures

Asynchronous

Synchronous, with faults!

The Byzantine Empire (circa 2009 AD)

Synchronous, with or without failures

Asynchronous

Recasting the problem

- Misguided
- Maximize performance when
  - the network is synchronous
  - Dangerous and servers behave correctly
- While remaining
  - Futile if at most f servers fail
  - eventually live

Recasting the problem

- Misguided
  - it encourages systems that fail to deliver BFT
- Dangerous
- Futile

Recasting the problem

- Misguided
  - it encourages systems that fail to deliver BFT
- Dangerous
  - it encourages fragile optimizations
- Futile

Recasting the problem

- Misguided
  - it encourages systems that fail to deliver BFT
- Dangerous
  - it encourages fragile optimizations
- Futile
  - it yields diminishing return on common case

BFT: a blueprint

- Build the system around execution path that:
  - provides acceptable performance across the broadest set of executions
  - it is easy to implement
  - it is robust against Byzantine attempts to push the system away from it

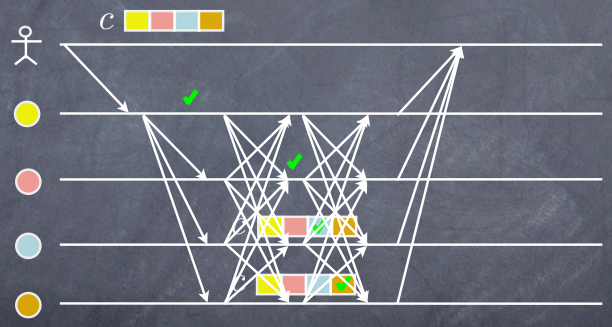
Revisiting conventional wisdom

- Signatures are expensive - use MACs
- View changes are to be avoided
- Hardware multicast is a boon

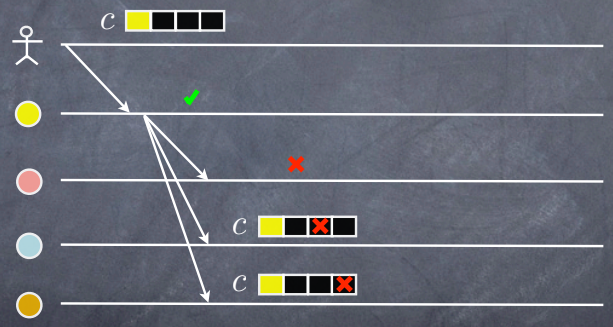
Revisiting conventional wisdom

- Signatures are expensive - use MACs
  - Faulty clients can use MACs to generate ambiguity
- View changes are to be avoided
- Hardware multicast is a boon

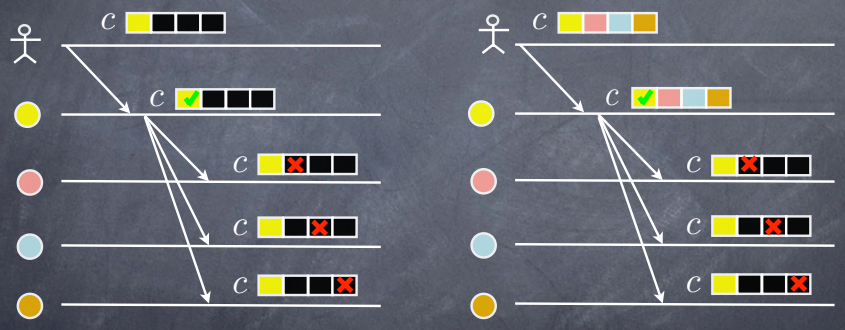
Big MAC Attack



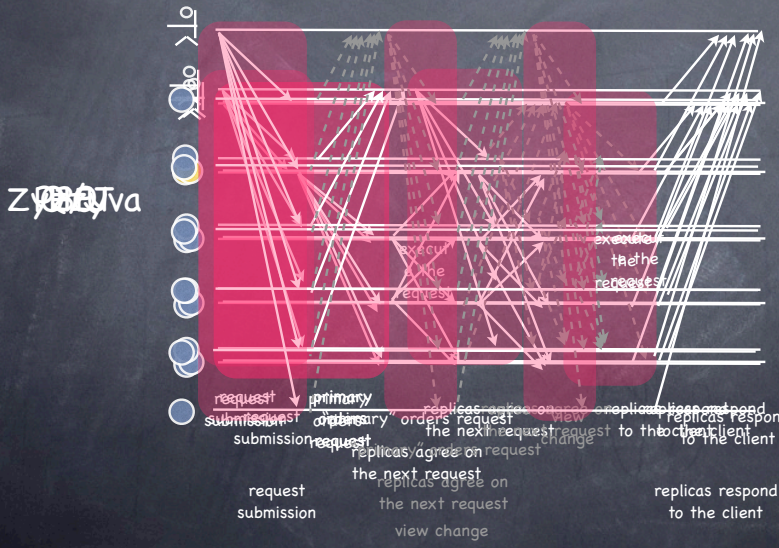
Big MAC Attack



Big MAC Attack



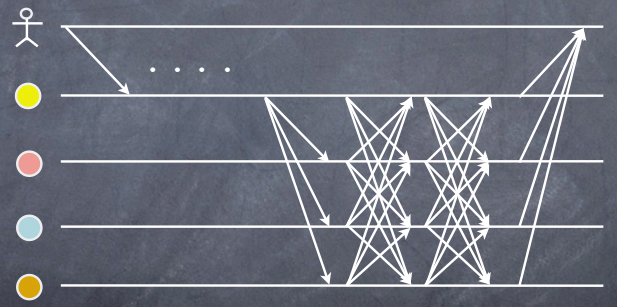
Big MAC Attack



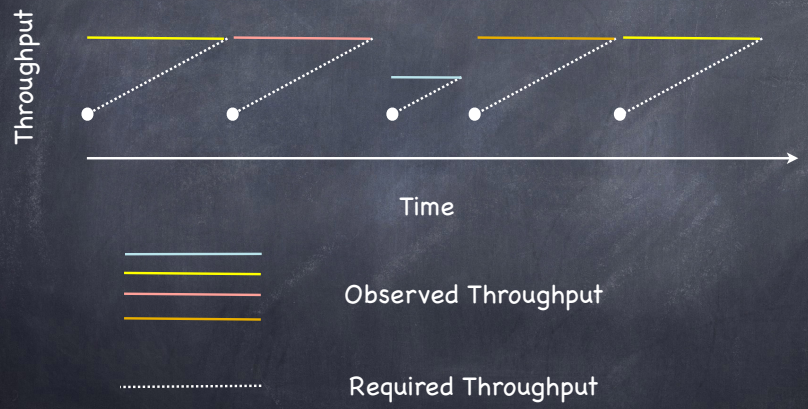
Revisiting conventional wisdom

- Signatures are expensive - use MACs
  - Faulty clients can use MACs to generate ambiguity
  - Aardvark requires clients to sign requests
- View changes are to be avoided
- Hardware multicast is a boon

Slow Primary



Adaptive View Changes



Revisiting conventional wisdom

- Signatures are expensive – use MACs
  - Faulty clients can use MACs to generate ambiguity
    - Aardvark requires clients to sign requests
- View changes are to be avoided
    - Aardvark uses regular view changes to maintain high throughput despite faulty primaries
- Hardware multicast is a boon

Revisiting conventional wisdom

- Signatures are expensive – use MACs
  - Faulty clients can use MACs to generate ambiguity
    - Aardvark requires clients to sign requests
- View changes are to be avoided
    - Aardvark uses regular view changes to maintain high throughput despite faulty primaries
- Hardware multicast is a boon
    - Aardvark uses separate work queues for clients and individual replicas

Throughput

	Best case	Faulty Client	Client Flood	Faulty Primary	Faulty Replica
PBFT	62K	0	crash	1k	250
QU	24K	0	crash	NA	19K
HQ	15K	NA	4.5K	NA	crash
Zyzzyva	80K	0	crash	crash	0
Aardvark	39K	39K	7.8K	37K	11K