BYZANTINE FAULT TOLERANT SOFTWARE-DEFINED NETWORKING (SDN) CONTROLLERS

KARIM ELDEFRAWY* AND TYLER KACZMAREK**

* INFORMATION AND SYSTEMS SCIENCES LAB (ISSL), HRL LABORATORIES
** UNIVERSITY OF CALIFORNIA IRVINE (UCI), WORK CONDUCTED WHILE AT HRL.

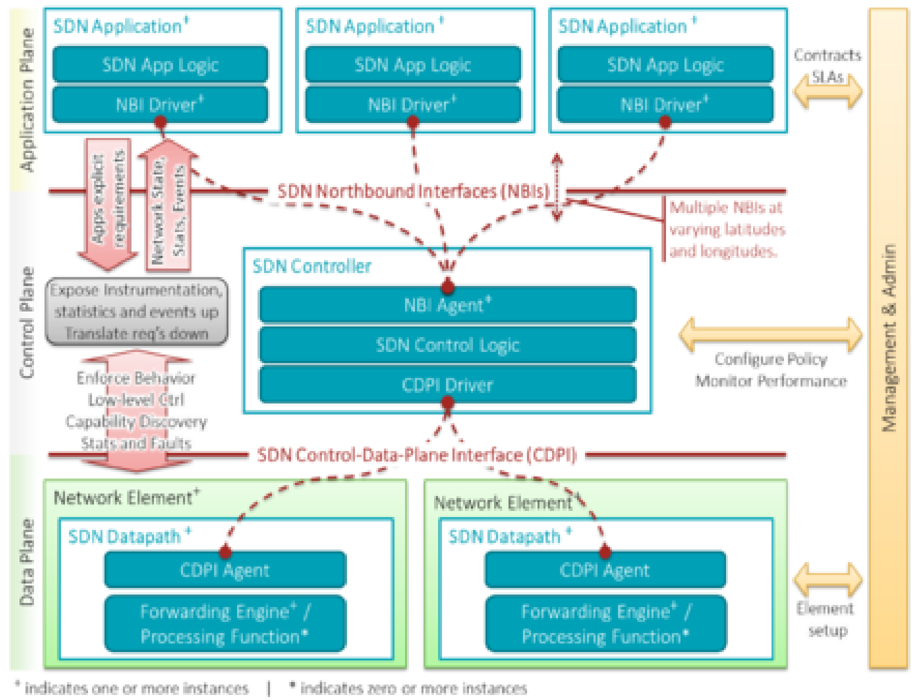
KMELDEFRAWY@HRL.COM
TKACZMAR@UCI.EDU

Overview of Talk

- Introduction and Motivation
- SDN Tools
- BFT-SMaRT Consensus Protocol
- BFT SDN System Design
- BFT SDN Controller Prototype Performance Results
- Improvements and Future Direction

Software-Defined Networking (SDN)

- **Network Function Virtualization (NFV)**
- **Decouples Forwarding Operations from Control Decisions**
- **Introduces centralized controller**
  - **Controller dictates forwarding rules**
  - **Routers act as dumb switches**
- **Openflow emerged as de facto SDN protocol**



Typical SDN Architecture
(Wikipedia, 2016)

Openflow Protocol: Messages

- **Openflow introduces unique message types for SDN control**
  - **PACKET_IN**
    - **Switch sends captured packet to controller on 2 occasions**
      - Miss in the flow table
      - Explicitly specified as per match rule
  - **FLOW_MOD**
    - **Controller modifies state of switch**
  - **PACKET_OUT**
    - **Controller directly injects packet into switch's data plane**

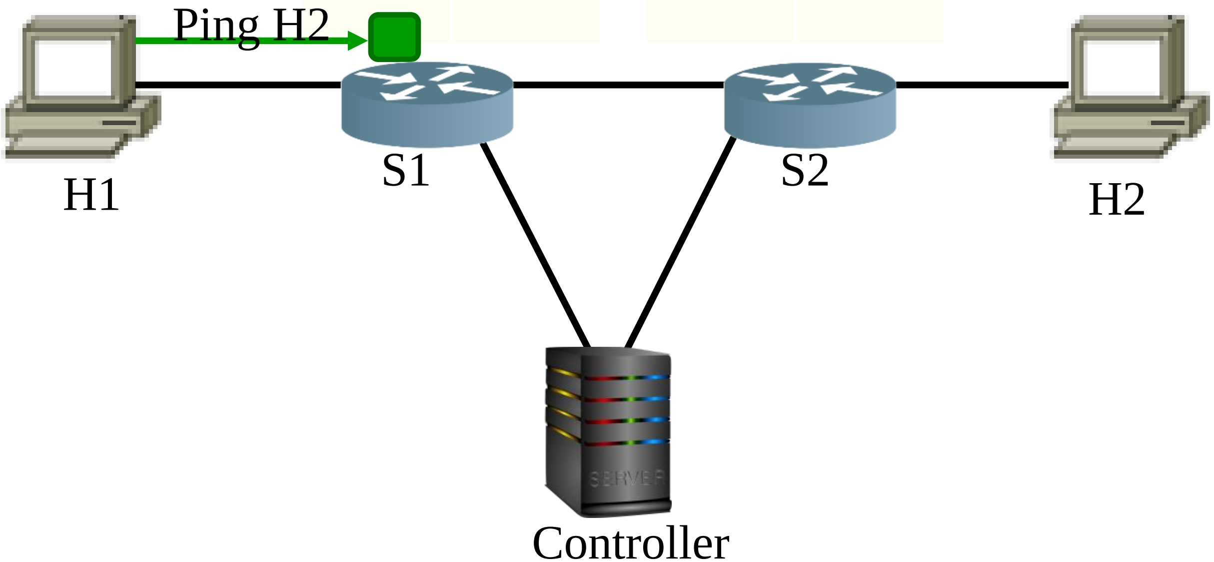
Openflow Protocol: Installing a Flow

S1's Flow Table

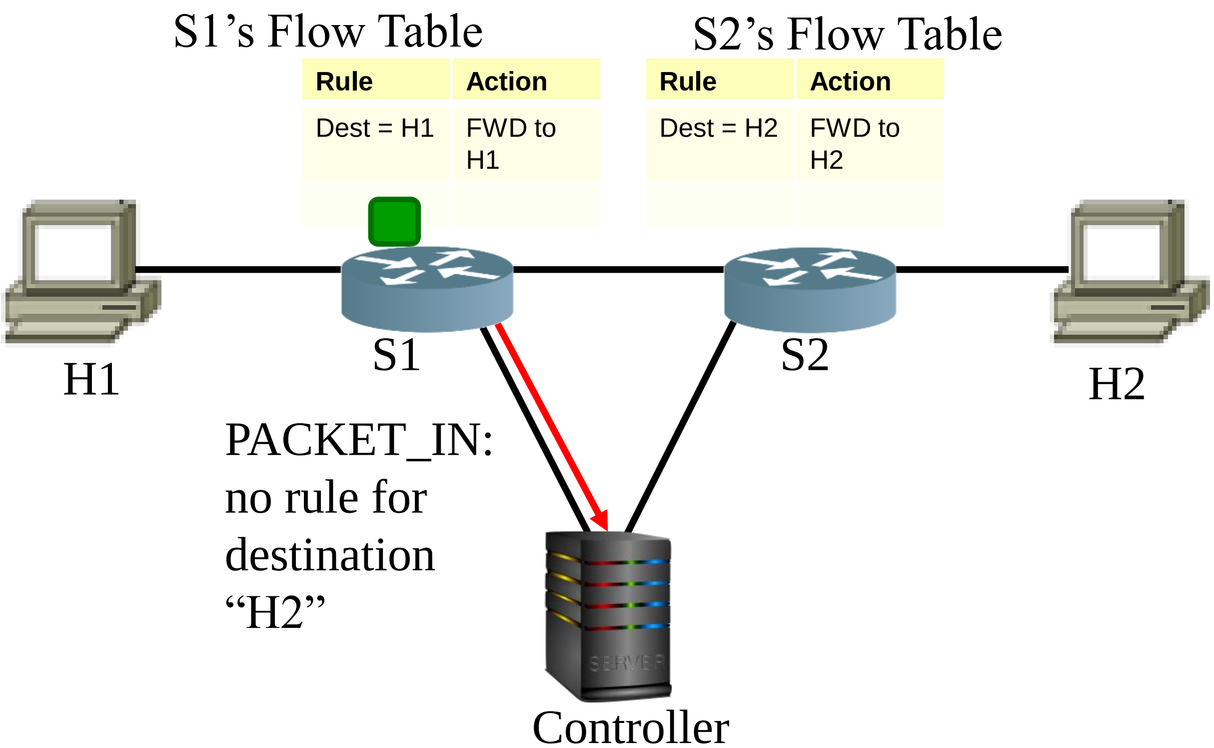
Rule	Action
Dest = H1	FWD to H1

S2's Flow Table

Rule	Action
Dest = H2	FWD to H2



Openflow Protocol: Installing a Flow



Rule	Action
Dest = H1	FWD to H1

Rule	Action
Dest = H2	FWD to H2

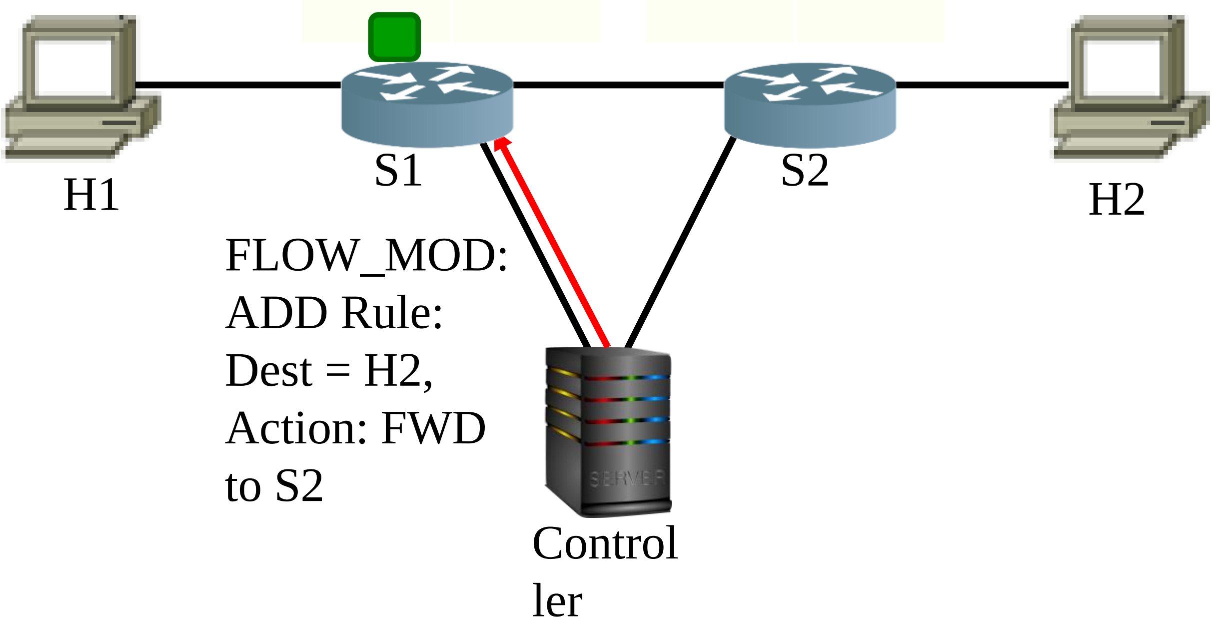
Openflow Protocol: Installing a Flow

S1's Flow Table

Rule	Action
Dest = H1	FWD to H1

S2's Flow Table

Rule	Action
Dest = H2	FWD to H2



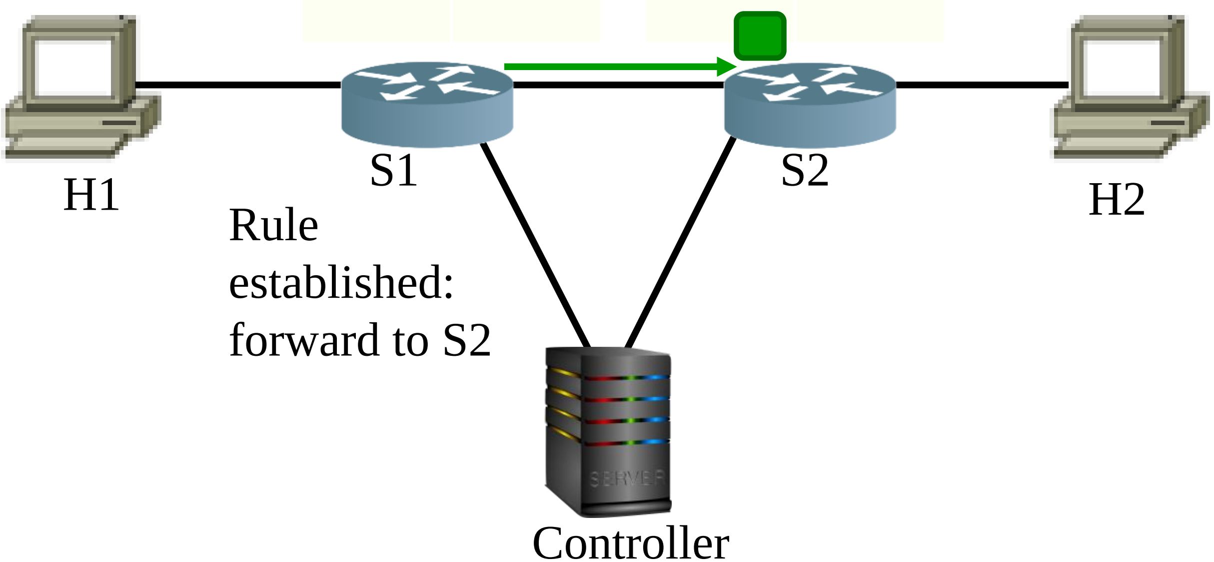
Openflow Protocol: Installing a Flow

S1’s Flow Table

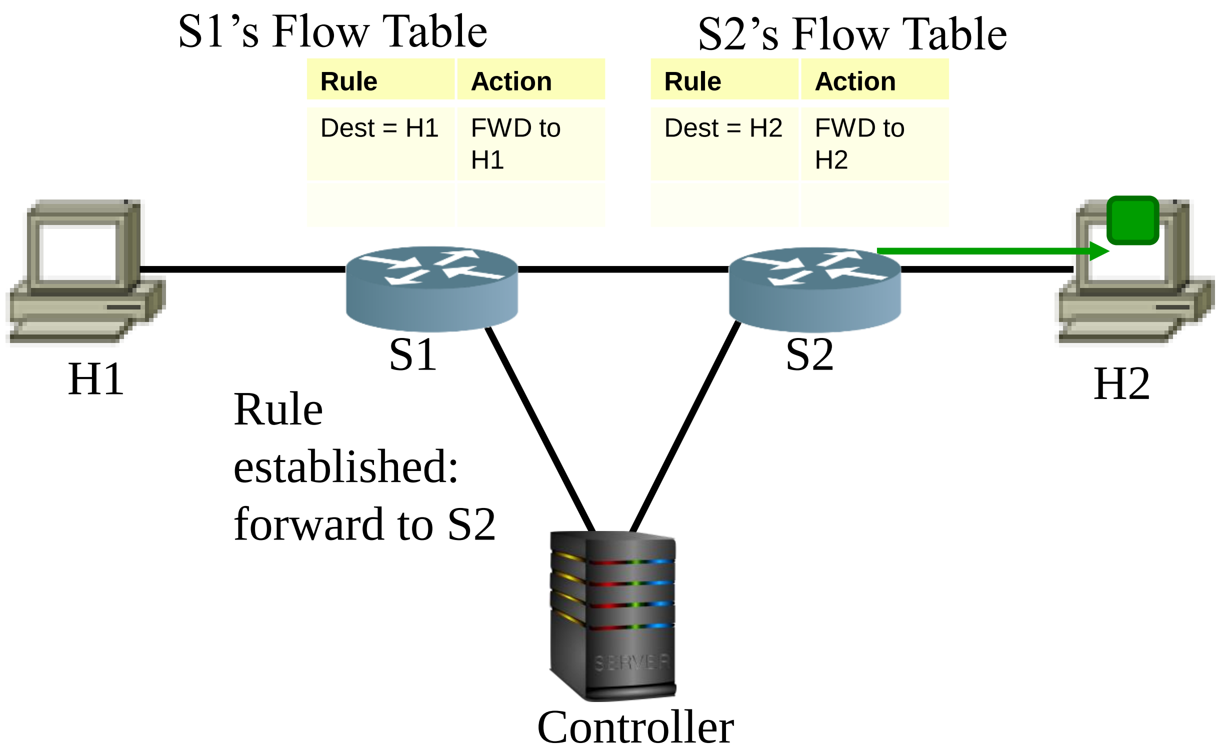
Rule	Action
Dest = H1	FWD to H1

S2’s Flow Table

Rule	Action
Dest = H2	FWD to H2



Openflow Protocol: Installing a Flow



Our Motivation

- **SDN architecture introduces single point of failure**
- **Unique threats facing SDN (Kreutz et. al 2013)**
  - **Attacks on Control Plane**
  - **Attacks on Controller Vulnerabilities**
  - **Few mechanisms ensuring trust between controller and management applications**
- **Current implementations do not account for failing or corrupted controllers**

Our Contributions

- **Designed and prototyped a Byzantine Fault Tolerant (BFT) SDN controller**
  - **Able to handle *f (1)* corrupted controller out of *3f+1 (4)* replica instances**
  - **Integrated into both OpenFlowJ and Beacon SDN controllers**
  - **Relies on BFT state machine replication (BFT-SMaRT)**

OpenflowJ and Beacon

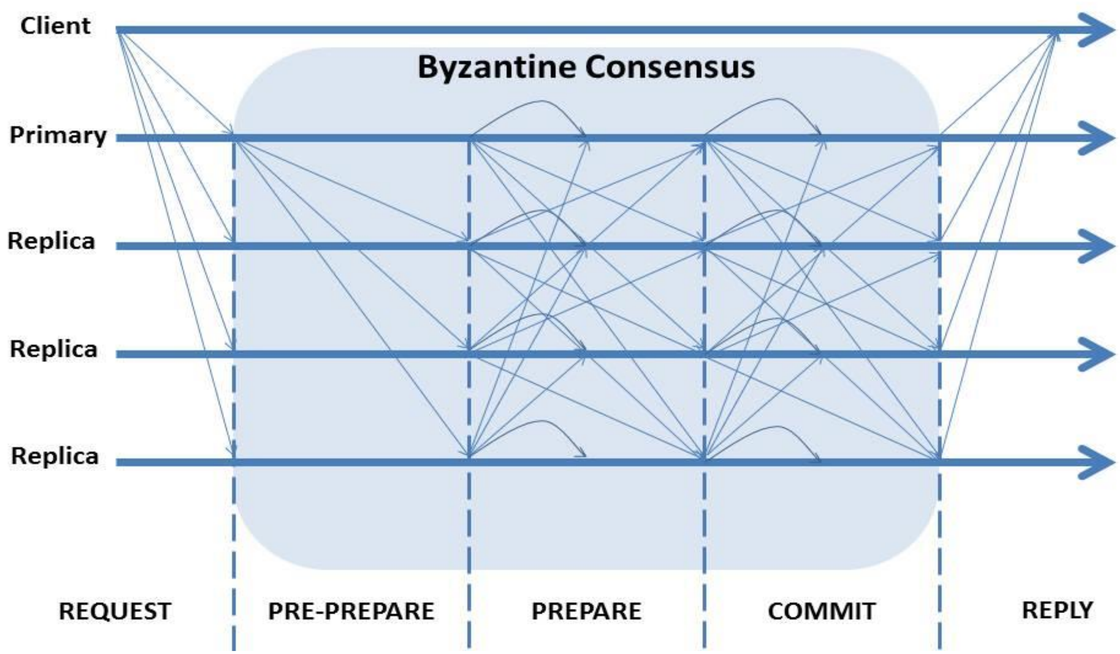
- OpenflowJ
  - Basic controller implementation
  - Close implementation of Openflow SDN standard
  - Java-based SDN controller

- Beacon
  - Optimizations on top of OpenflowJ
    - Leverages Parallelism
    - Modular
    - Extensible

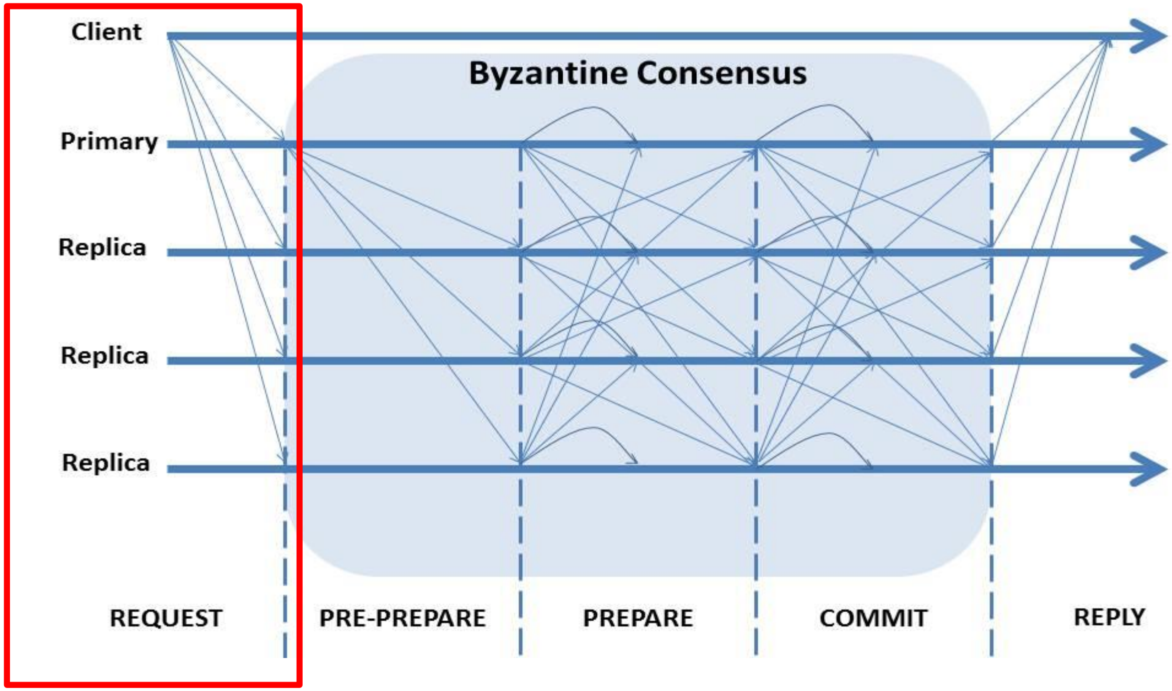
BFT-SMaRT

- Developed by Bessani et. al, (Technical Report, 2013)
- Implemented in Java (https://github.com/bft-smart/library)
- Close implementation of Lamport's original schema
- Accepts *f* faults with *3f+1* total replicas
- Accepts batching
- Used for base-level implementation

BFT-SMaRT Consensus Protocol

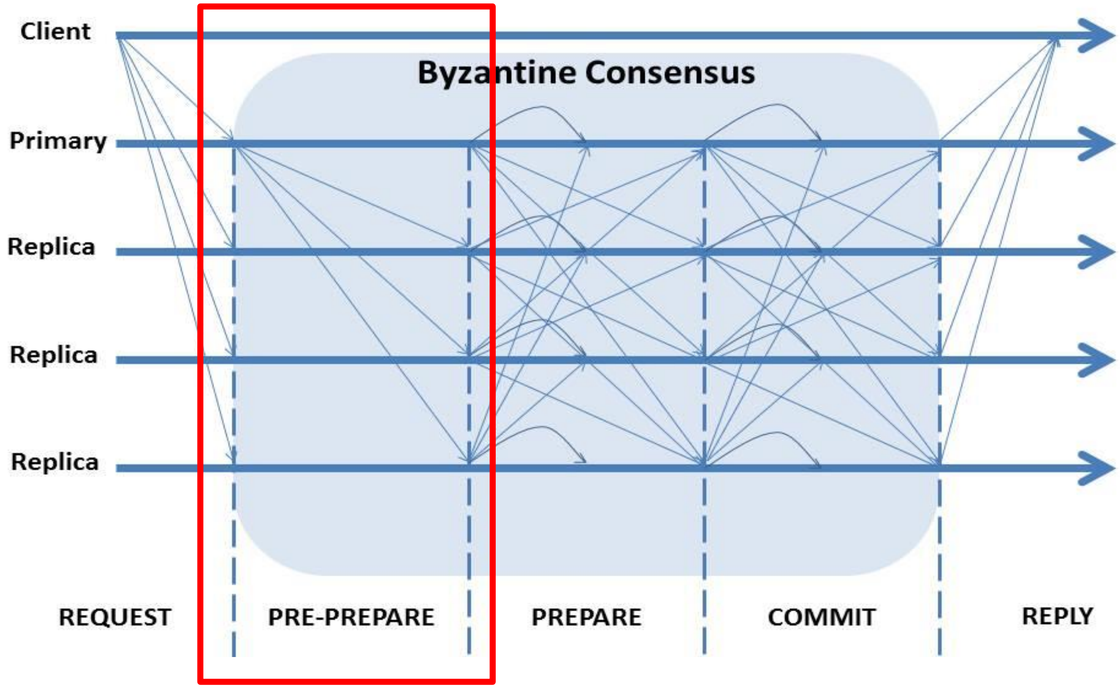


BFT-SMaRT Consensus Protocol - Request



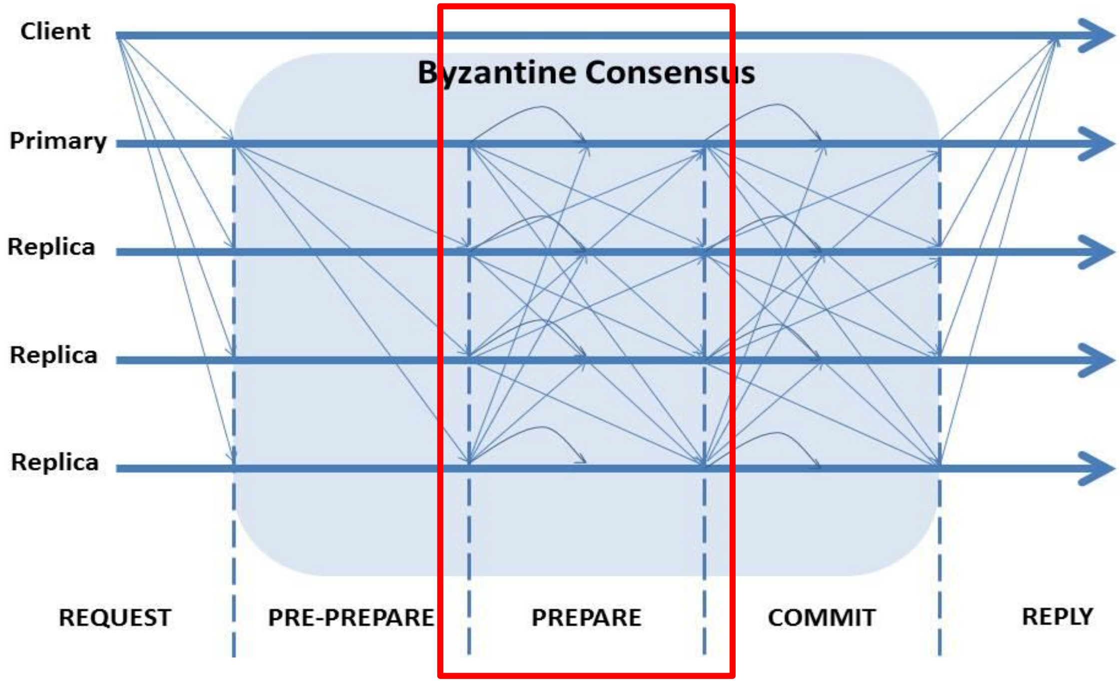
- **Request – Client multicasts request**

BFT-SMaRT Consensus Protocol – Pre-Prepare



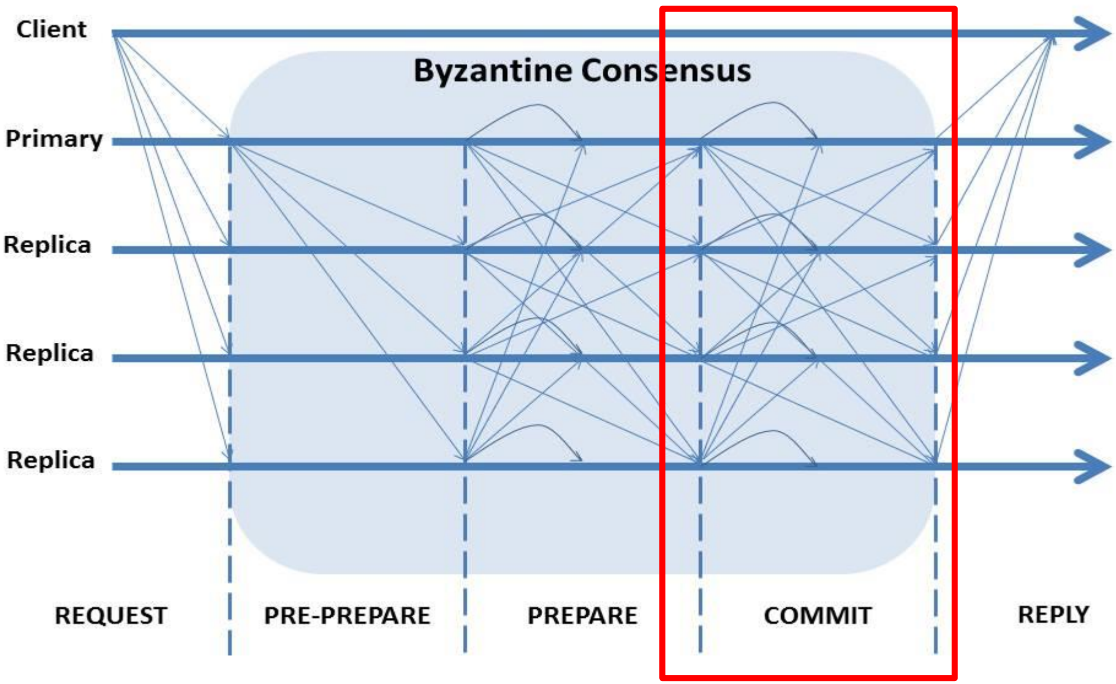
- **Pre-prepare – Primary orders request and multicasts unique number**

BFT-SMaRT Consensus Protocol - Prepare



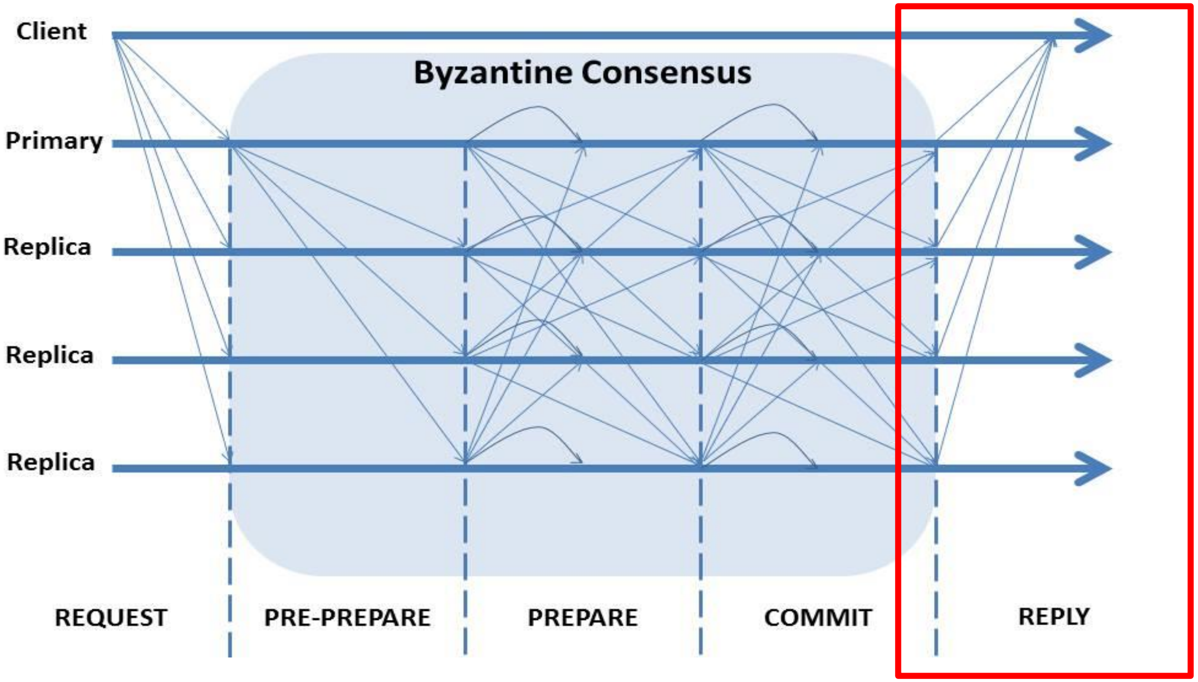
- **Prepare – All replica multicast out Prepare**

BFT-SMaRT Consensus Protocol - Commit



- **Commit –Replicas multicast Commit; request added to committed certificate**

BFT-SMaRT Consensus Protocol - Reply



- **Reply –Replicas compute request and reply to client, result accepted once *f+1* identical responses received**

BFT Switch Design

- **Based on Open vSwitch**
- **Same switch design used for both controller designs**
- **Introduced "Client Proxy" to interface with BFT Controllers**
  - **Switch sends PACKET_IN requests to Proxy, which formats into REQUEST messages as per BFT-SMaRT**
  - **Client Proxy formats RESPONSE messages from BFT controller into either PACKET_OUT or FLOW_MOD messages**

BFT Controller Design – SimpleBFT

- Extension of OpenflowJ control logic
  - Responds to individual PACKET_IN messages
  - Issues identical PACKET_OUT and FLOW_MOD responses to OpenFlowJ

- Interfaces with Client Proxy instead of directly with switch
  - Reaches consensus on each given message
  - Primary changes after primary found faulty

BFT Controller Design – BeaconBFT

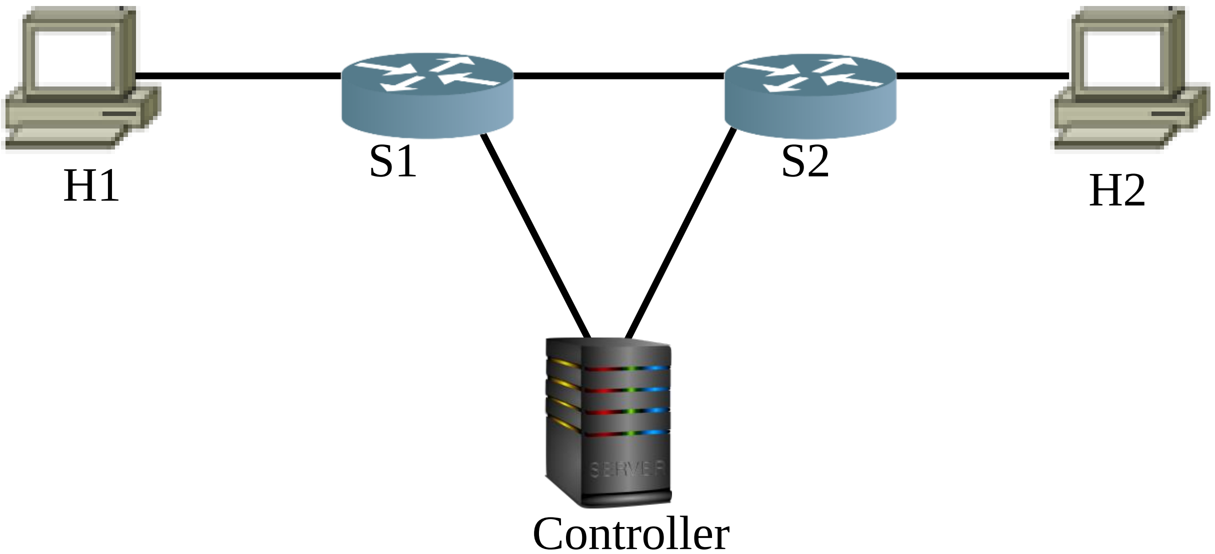
- Identical modifications made to Beacon as OpenflowJ

- Extension of Beacon control logic
  - Responds to individual PACKET_IN messages
  - Issues identical PACKET_OUT and FLOW_MOD responses to OpenFlowJ

- Interfaces with Client Proxy instead of directly with switch
  - Reaches consensus on each given message
  - Primary changes after primary found faulty

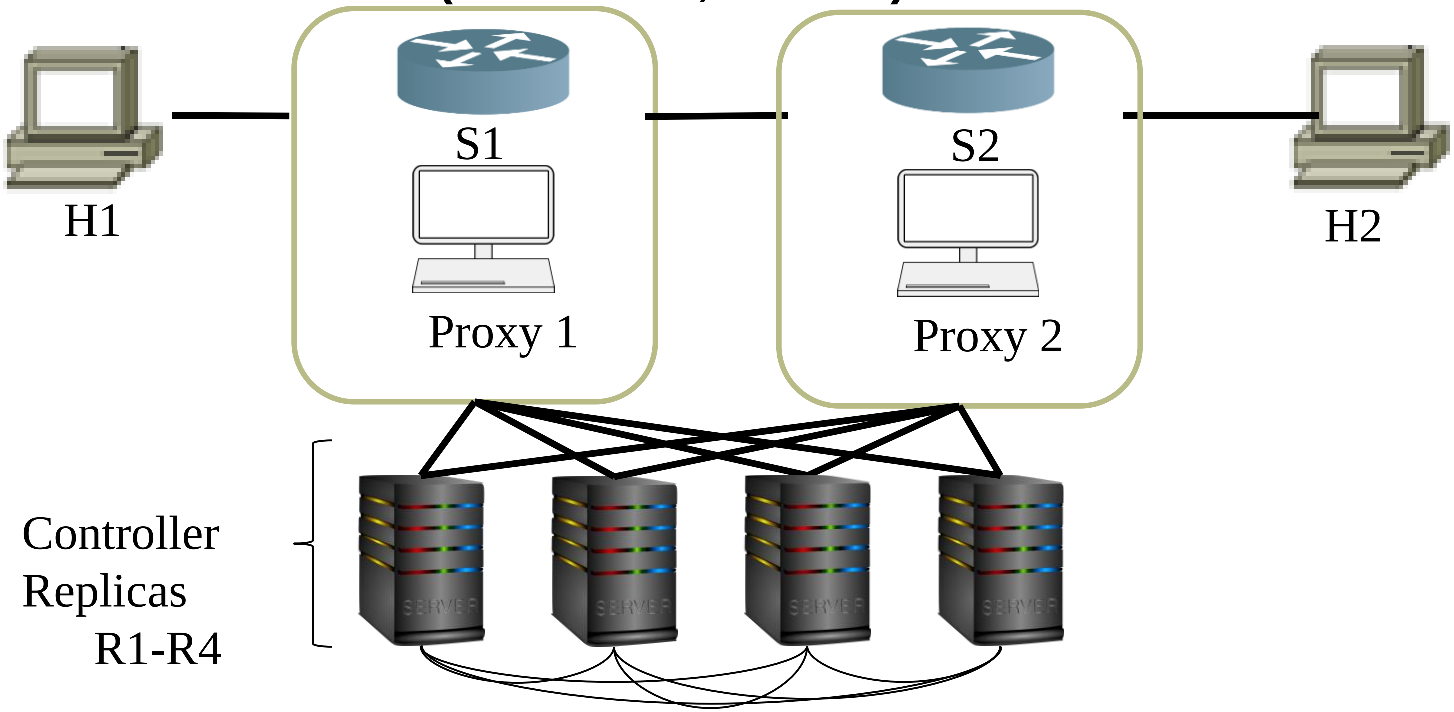
BFT Controller Design - Architecture Changes

- **Typical Architecture (2 switches, 2 hosts)**



BFT Controller Design – Architecture Changes

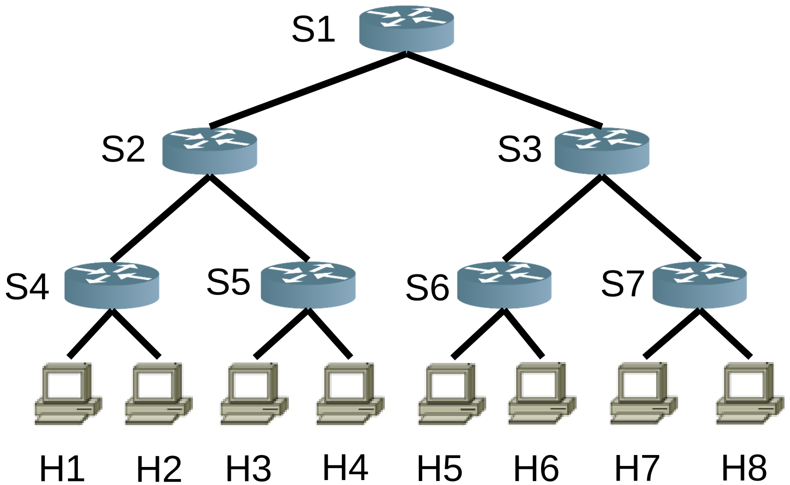
- **Modified Architecture (2 switches, 2 hosts)**



Performance Testing

- **Tested in Cbench on an HP EliteBook 8560W**
  - **Windows 7, 64 bit**
  - **4096MB RAM**
  - **Intel Core i7**
    - **2.7GHz**
    - **4 CPUs**

- **Simulated binary tree network of depth 5 in mininet**



Example Binary Tree topology of depth 3

Performance Results

Controller	End-to-End Setup Duration	Flow Setup Delay	Flow Setup Rate (flow mods/second)
OpenflowJ	376ms	9.44ms	106.9 fm/sec
SimpleBFT	775ms	31.7ms	59.3 fm/sec
Beacon	77ms	0.5ms	550.6 fm/sec
BeaconBFT	475ms	14.5ms	87.0 fm/sec

Performance Issues

- Slowdown for OpenFlowJ much smaller than for Beacon
- Beacon leverages parallelism for performance gains
- BeaconBFT unable to leverage parallelism
  - Total ordering required for BFT operation

Speeding up the Switch

- Batch Client requests
  - Minimize overhead per PACKET_IN message
- Integrate BFT Switch with client
  - Removes reliance on proxy for communication
  - 1 less step in communication

Speeding up the Controller

- **Alternative BFT protocols**
  - **Naïve implementation maximizes communication**
- **Spinning BFT**
  - **Allows multiple views**
  - **Greater parallelism support**
- **Speculative BFT**
  - ***f+1* replicas running until fault occurs**
  - **Reduces messages needed by a factor of 3 in PREPARE and COMMIT steps**

Speeding up the Network

- **Scalability issues even with non-BFT SDN controllers in reactive mode**
- **Convert to proactive controller**
  - **Greatest costs are in pre-computing many flows**
  - **Minimize active PACKET_IN messages generated**
  - **Less real-time computation**

LABORATORIES

Questions?