Cristina Nita-Rotaru

CS6740: Network security

DHT; SDN.

What is network security

``Network Security is the process of taking physical and software preventative measures to protect the underlying networking infrastructure from unauthorized access, misuse, malfunction, modification, destruction, or improper disclosure, thereby creating a secure platform for computers, users and programs to perform their permitted critical functions within a secure environment.''

SANS Institute

- Goals for protocols
- Attackers trying to disrupt those goals
- Attackers have resources and means – attacker model

- Security means – something being possible/impossible under certain assumptions about the attacker and its resources and means, and about the boundary of interaction of the target-system with ``the rest of the world".

The learning does not end here

- Some stacks not likely to change anytime soon – the Internet. But new services will bring even more complex interactions
- Stay skeptical and pay attention to foundations: if something sounds too good to be true, it probably is not as good
- Keep up to date with more attacks and new crypto developments and bounds
- Focus also on design not only attacks
- Try to go to the root/essence of problem or solutions

2: DHT - Chord

CHORD

- Efficient lookup of a node which stores data items for a particular search key.
- Provides only one operation: given a key, it maps the key onto a node.
- Example applications:
  - Co-operative mirroring
  - Time-shared storage
  - Distributed indexes
  - Large-scale combinatorial search

Design Goals

- Load balance: distributed hash function, spreading keys evenly over nodes
- Decentralization: CHORD is fully distributed, nodes have symmetric functionality, improves robustness
- Scalability: logarithmic growth of lookup costs with number of nodes in network
- Availability: CHORD guarantees correctness, it automatically adjusts its internal tables to ensure that the node responsible for a key can always be found

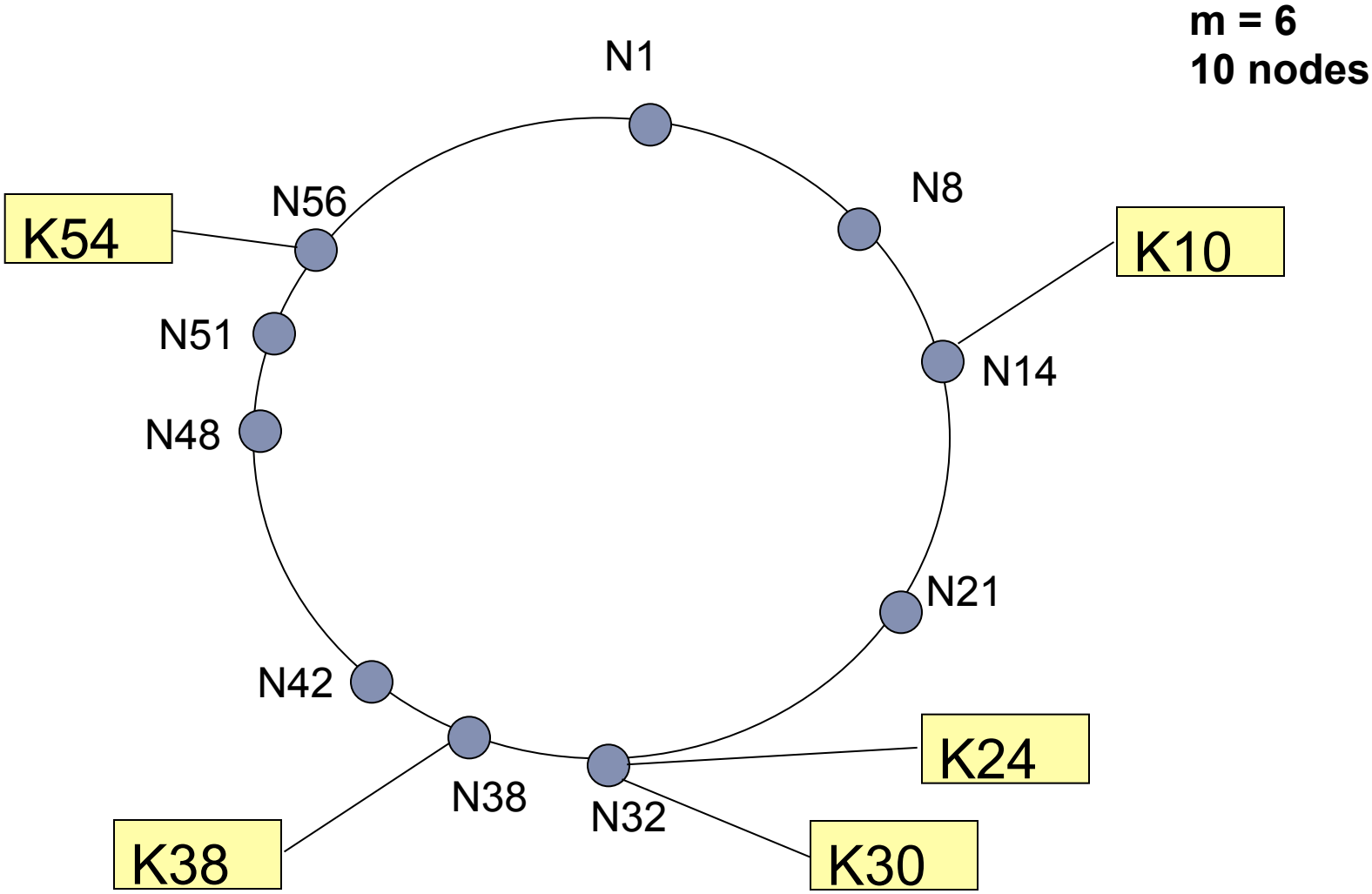
Assumptions

- Communication in underlying network is both symmetric and transitive
- Assigns keys to nodes with consistent hashing
- Hash function balances the load
- Participants are correct, nodes can join and leave at any time
- Nodes can fail

Chord Rings

- Key identifier = SHA-1(key)
- Node identifier = SHA-1(IP address)
- Consistent hashing function assigns each node and key an m-bit identifier using SHA-1
- Mapping key identifiers to node identifiers:
  - Identifiers are ordered on a circle modulo 2^m called a chord ring.
  - The circle is split into contiguous segments whose endpoints are the node identifiers. If i1 and i2 are two adjacent IDs, then the node with ID greater identifier i2 owns all the keys that fall between i1 and i2.

Example of Key Partitioning in Chord

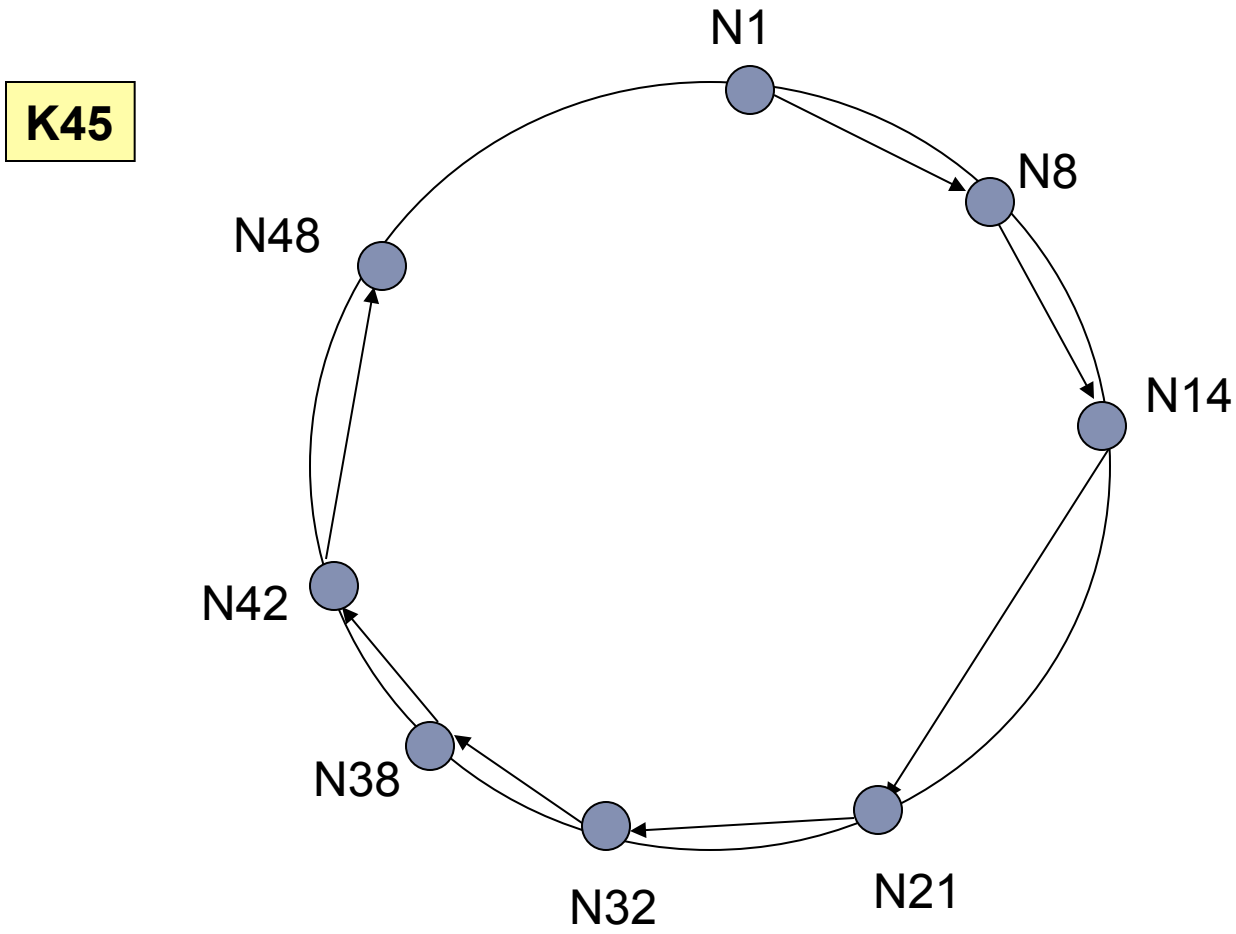


How to Perform Key Lookup

- Assume that each node knows only how to contact its current successor node on the identifier circle, then all node can be visited in linear order.
- When performing a search, the query for a given identifier could be passed around the circle via these successor pointers until they encounter the node that contains the key corresponding to the search.

Example of Key Lookup Scheme

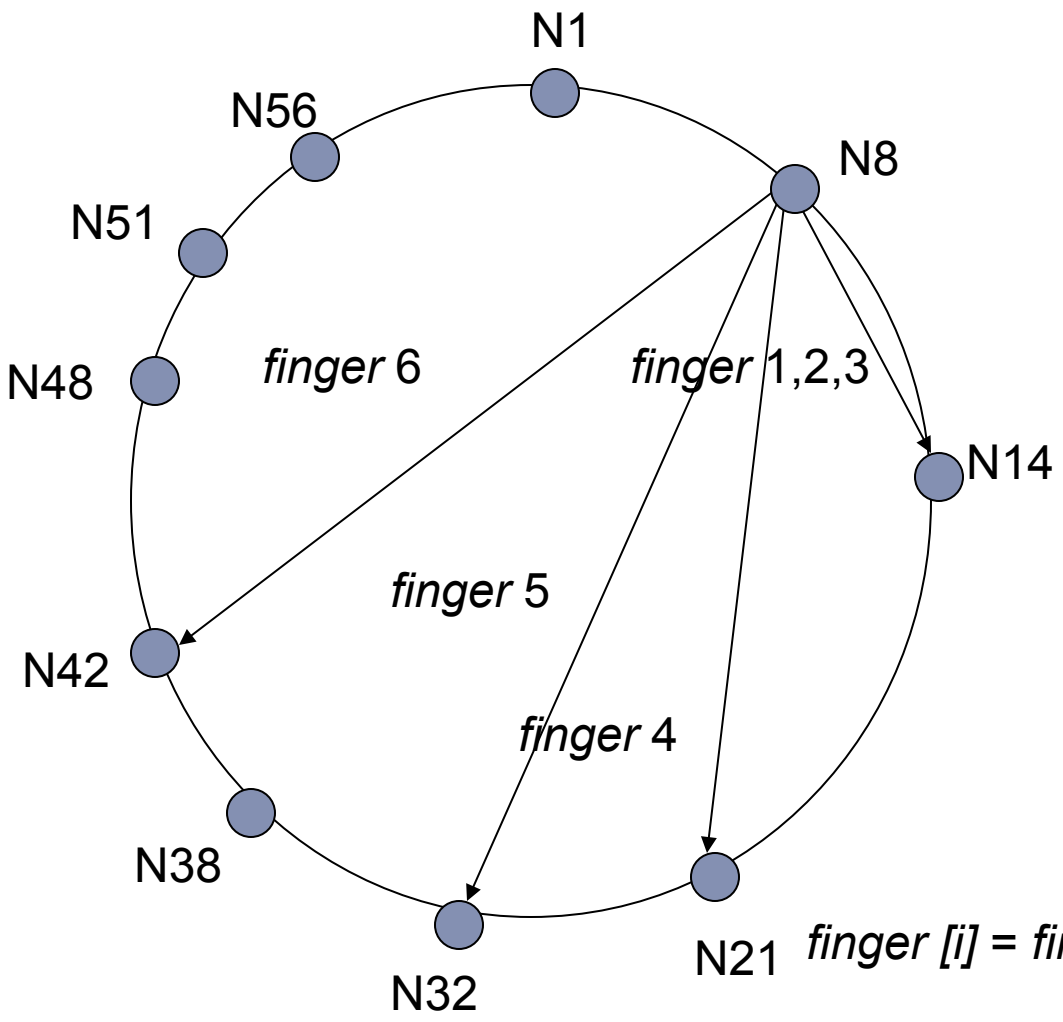
succesor(k) = first node whose ID is >= ID of k in identifier space



Scalable Key Location

- To accelerate lookups, Chord maintains additional routing information (m entries): finger table
- The i^{th} entry in the table at node n contains the identity of the first node s that succeeds n by at least 2^{i-1} on the identifier circle.
- $s = successor(n+2^{i-1})$.
- s is called the i^{th} finger of node n

Scalable Lookup Scheme



m = 6

Finger Table for N8

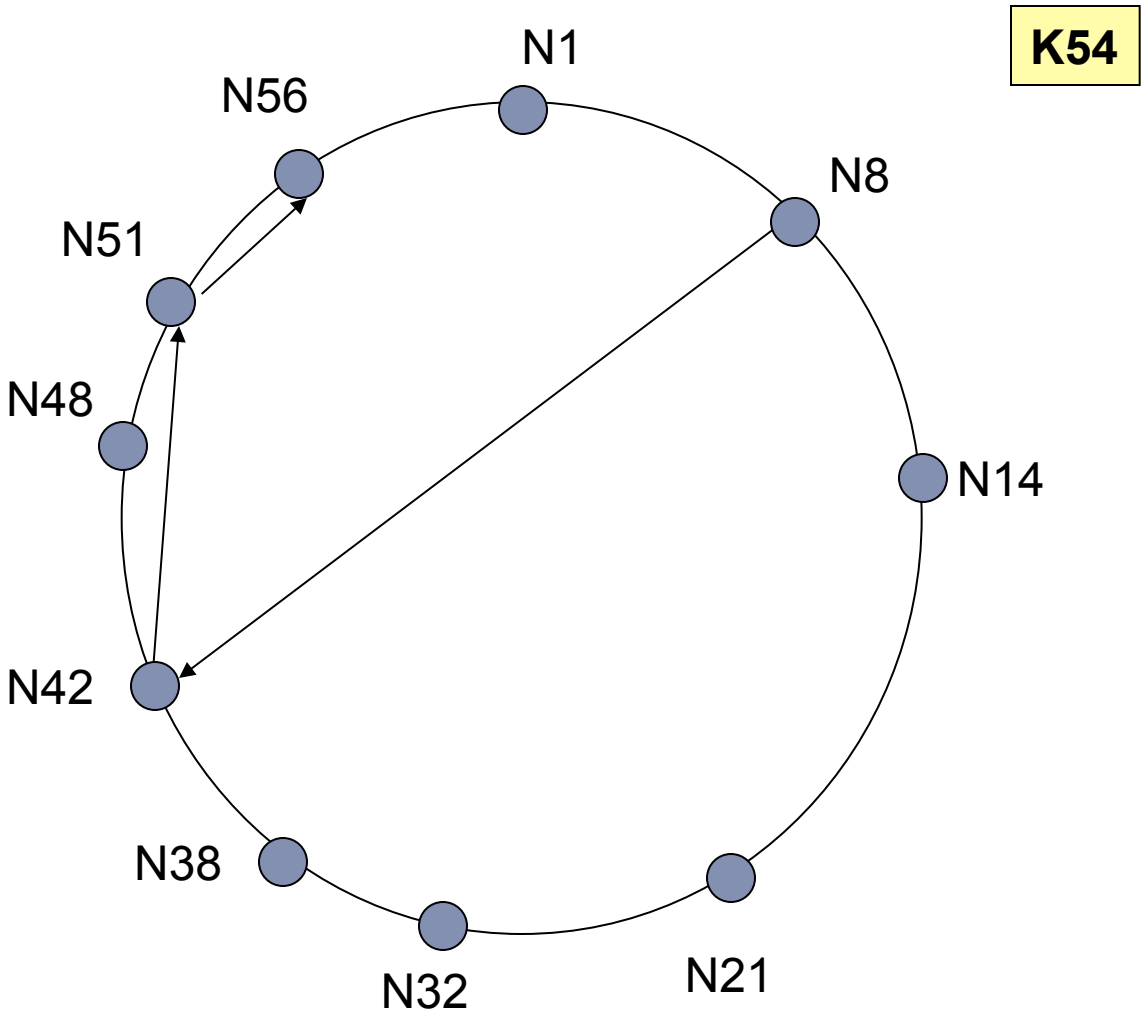
N8+1	N14
N8+2	N14
N8+4	N14
N8+8	N21
N8+16	N32
N8+32	N42

finger [i] = first node that succeeds $(n+2^{i-1}) \bmod 2^m$

Scalable Lookup

- Each node has finger entries at power of two intervals around the identifier circle
- Each node can forward a query at least halfway along the remaining distance between the node and the target identifier.

Lookup Using Finger Table



Node Joins and Failures/Leaves

- When a node N joins the network, some of the keys previously assigned to N's successor should become assigned to N.
- When node N leaves the network, all of its assigned keys should be reassigned to N's successor.
- How to deal with these cases?

Node Joins and Stabilizations

- Everything relies on successor pointer.
- Up to date successor pointer is sufficient to guarantee correctness of lookups
- Idea: run a "stabilization" protocol periodically in the background to update successor pointer and finger table.

Stabilization Protocol

- Guarantees to add nodes in a fashion to preserve reachability
- Does not address the cases when a Chord system thas split into multiple disjoint cycles, or a single cycle that loops multiple times around the identifier space

Stabilization Protocol (cont.)

- Each time node N runs stabilize protocol, it asks its successor for its predecessor p, and decides whether p should be N's successor instead.
- Stabilize protocol notifies node N's successor of N's existence, giving the successor the chance to change its predecessor to N.
- The successor does this only if it knows of no closer predecessor than N.

Impact of Node Joins on Lookups

- If finger table entries are current then lookup finds the correct successor in O(log N) steps
- If successor pointers are correct but finger tables are incorrect, correct lookup but slower
- If incorrect successor pointers, then lookup may fail

Voluntary Node Departures

- Leaving node may transfers all its keys to its successor
- Leaving node may notify its predecessor and successor about each other so that they can update their links

Node Failures

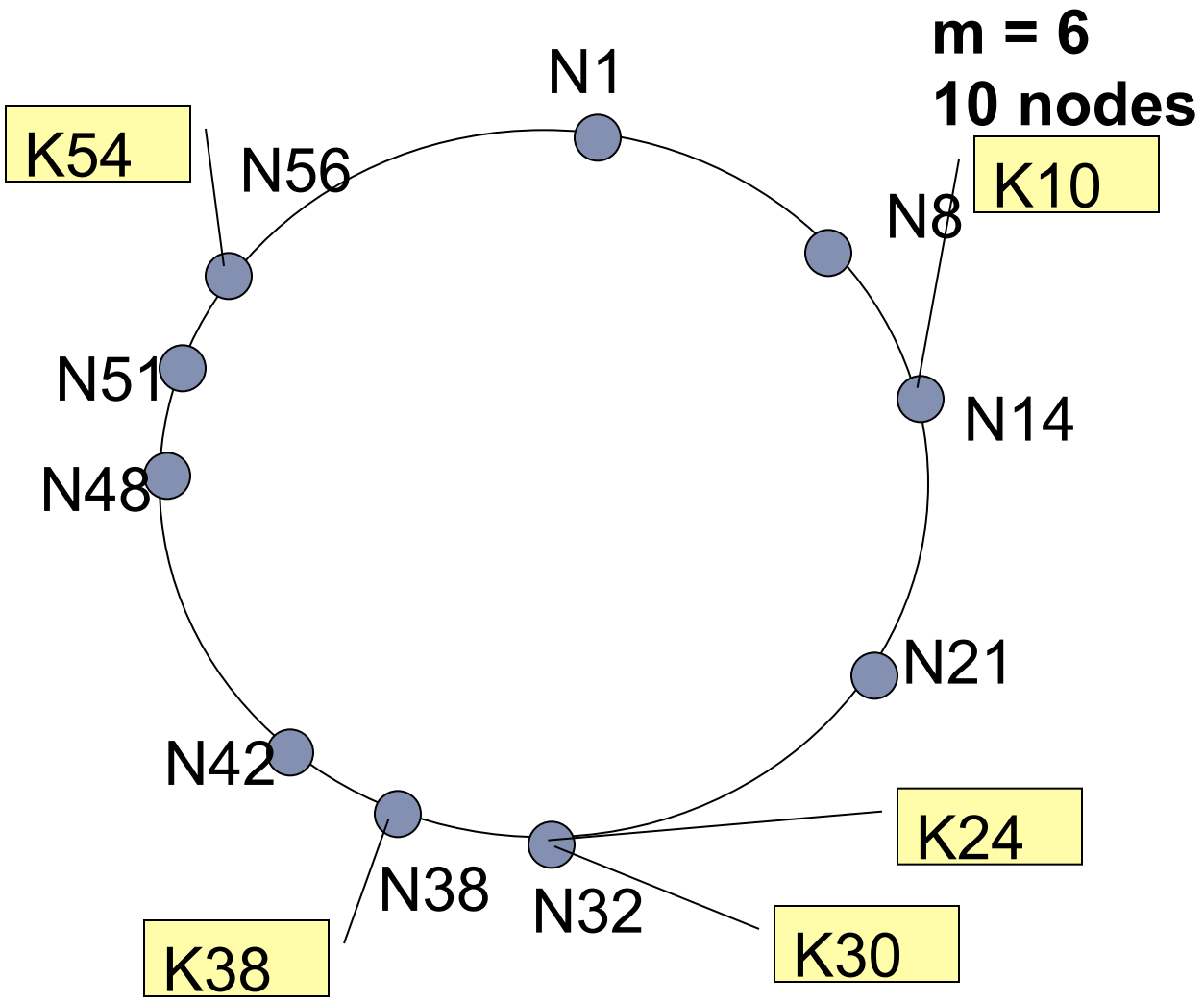
- Stabilize successor lists:
  - Node N reconciles its list with its successor S by copying S's successor list, removing its last entry, and prepending S to it.
  - If node N notices that its successor has failed, it replaces it with the first live entry in its successor list and reconciles its successor list with its new successor.

CHORD Summary

- Efficient location of the node that stores a desired data item is a fundamental problem in P2P networks
- Separates correctness (successor) from performance (finger table)
- Chord protocol solves it in a efficient decentralized manner
  - Routing information: O(log N) nodes
  - Lookup: O(log N) nodes
  - Update: O(log2 N) messages
- It also adapts dynamically to the topology changes introduced during the run

Node ID Assignment Implications

- Node ID determines where will the node be placed in the structured overlay
- Determines who the neighbors are going to be
- Determines what objects a node will hold



Attacks Based on Node ID Assignment

- What if attacker can choose the ID of a node?
  - Surround a victim node
  - Partition a p2p network
  - Can control what object will be a replica for
  - Holding objects allow an attacker to delete, corrupt or deny access to objects
- Can an attacker choose the ID of a node?
  - In some systems ID is randomly generated
  - In some systems ID is the hash of the IP address

Sybil Attack

- Attack particular to peer networks, a malicious attacker takes/ forges multiple identities
- Result: attacker controls a significant part of the system while correct nodes are not aware of this, they see different identities
  - destroy cohesion of the overlay
  - observe network status
  - slow down, destroy overlay
  - DoS
- How to ensure/validate distinct identities refer to distinct entities?

Evaluating Identity

- Straightforward form of identity: secure hash of a public key
- How to evaluate/learn the identity of other entities
  - Use a trusted agency (learn from trusted source)
  - A node has a direct way of validating other nodes - direct validation (learn directly)
  - Using other untrusted agencies - indirect validation (learn from others)
- Which one is best?

Direct and Indirect Validation (Untrusted Sources)

- Utilize computational tasks to validate distinctness;
  - validate distinctness of two entities by getting them to perform some task (for example a computational puzzle) that a single entity could not
  - can not assume homogeneous resources, only minimum; faulty entity could have more than minimum
  - The goal is to make it practical impossible for an adversary to have challenges issued simultaneously, limit the number of identities he can forge

Direct Validation Limitations

- Even with severely resource constrains, a faulty entity can counterfeit a constant number of identities
- Each correct entity must simultaneously validate all the identities it is presented otherwise a faulty entity can counterfeit an unbounded number of identities

Indirect Validation

- A sufficiently large set of faulty entities can counterfeit an unbounded number of identities
- All entities in the system must perform their identity validations concurrently, otherwise a faulty identity can counterfeit a constant number of multiple identities

Certified (Secure) NodeID Assignment

- Delegate ID generation to trusted CAs
- Bind IPs with nodeIds such that colluding attackers can not exchange certificates
- Nodes must pay for certificates to prevent attackers from buying many "correct" certificates
- Works for static IP addresses
- Does not solve all problems: what happens if the IP changes?
- What happens if the trusted CA is not available or can not be reached?

Certified (Secure) NodeID Assignment

- How about distributed ID generation with periodic renewal of distributed IDs
  - Addresses single point of failure
  - Requires techniques to moderate the rate at which attackers can acquire node IDs

Routing Table Maintenance

- Routing table contains information about where to 'look next'
- Table is updated based on information from other nodes

m = 6

Finger Table for N8

N8+1	N14
N8+2	N14
N8+4	N14
N8+8	N21
N8+16	N32
N8+32	N42

finger [k] = first node that succeeds (n·

Attack Against Maintaining Routing Table

- Attackers can easily supply 'malicious' updates or can return incorrect lookup
  - point to faulty or non-existent nodes
  - fake the closest node
  - lie about next hop
- Result: lookup will fail (denial of information to a node) or the lookup algorithm will have sub-optimal performance

Secure Routing Table Maintenance

- Constrained Routing Tables: Identify invariants in the system and look for violations of the invariants
- Maintain two routing tables - one that uses proximity information and one that constrains entries to "specific" values
  - Proximity routing used in normal operation
  - Constrained routing used when failures occur:
- Other proposed solutions involve anonymous auditing

Secure Bootstrapping

- How to securely bootstrap the routing table?
  - A new node, n, picks a subset of bootstrap nodes to query and join the network
  - n uses the bootstrap information to initialize its constrained routing table

Attacks on Forwarding

- Simply ignore forwarding messages, route to the wrong node
  - Failed if ANY one in routing is faulty
  - Probability of routing successfully to a replica root is $(1-f)^{h-1}$
  - h is the number of average hops for delivering a message
  - h depends on the overlay

Secure Message Forwarding

- Ensure that with high probability, at least on copy of a message reaches every correct replica root
- Collect the prospective set of replica roots from the prospective root node
- Apply **routing failure test** to determine if routing worked correctly, If no, use ***redundant* and/or *iterative routing*.**

Testing Routing

- Route Failure Test:
  - Average density of nodeIds per unit of "volume" in the id space is greater than the average density of faulty nodes
  - Compares density of nodeIds in the neighbor set of the sender with the density of nodeIds close to the replica roots of the destination key
  - Have sender contact all prospective roots
  - Timeout to detect ignoring routing msgs, selecting the appropriate threshold not easy
- Use redundant routing when test fails
  - Neighbor set anycast - sends copies of message towards destination until they reach a node with the key in its neighbor set.
- How about false positives and false negatives when performing the routing failure test?
- Redundant routing has high overhead?

Iterative Routing

- Alternative to redundant routing
- Every lookup answer goes back to the requester that can verify that the next hop gets him closer (using the distance function) to the node hosting the object associated with the requested key
- Iterative routing is more secure, but more expensive

What Does Secure Routing Buy Us?

- Prevents attacks at join time: secure nodeID assignment and bootstrapping
- Ensure that when a correct node sends a message for a particular key, the message reaches all correct replica roots for the key with very high probability.
- What about the data? We need other mechanisms, for example self-certifying data

Self-Certifying Data

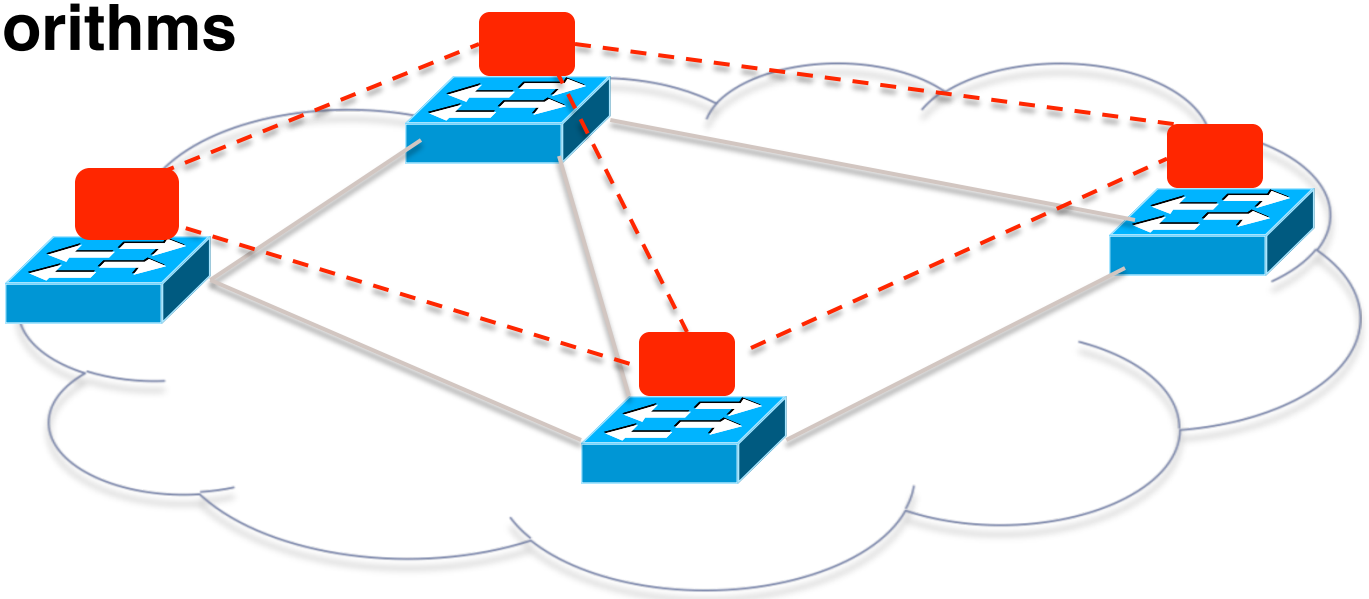
- Client can check data and only needs to rely on routing when certification check fails.
- Reduces the reliance on the redundant, secure routing primitive (you still need secure forwarding otherwise there is no data to verify in the first place)
- Uses concepts like proactive signature sharing or group keys/ signatures.
- Self-certifying data can eliminate the overhead of secure routing in common cases

3: SDN;OpenFlow

Slides for overview of SDN by Jennifer Rexford
Slide for challenges in SDN Theophilus Benson

Traditional Computer Networks

Control plane:
Distributed algorithms

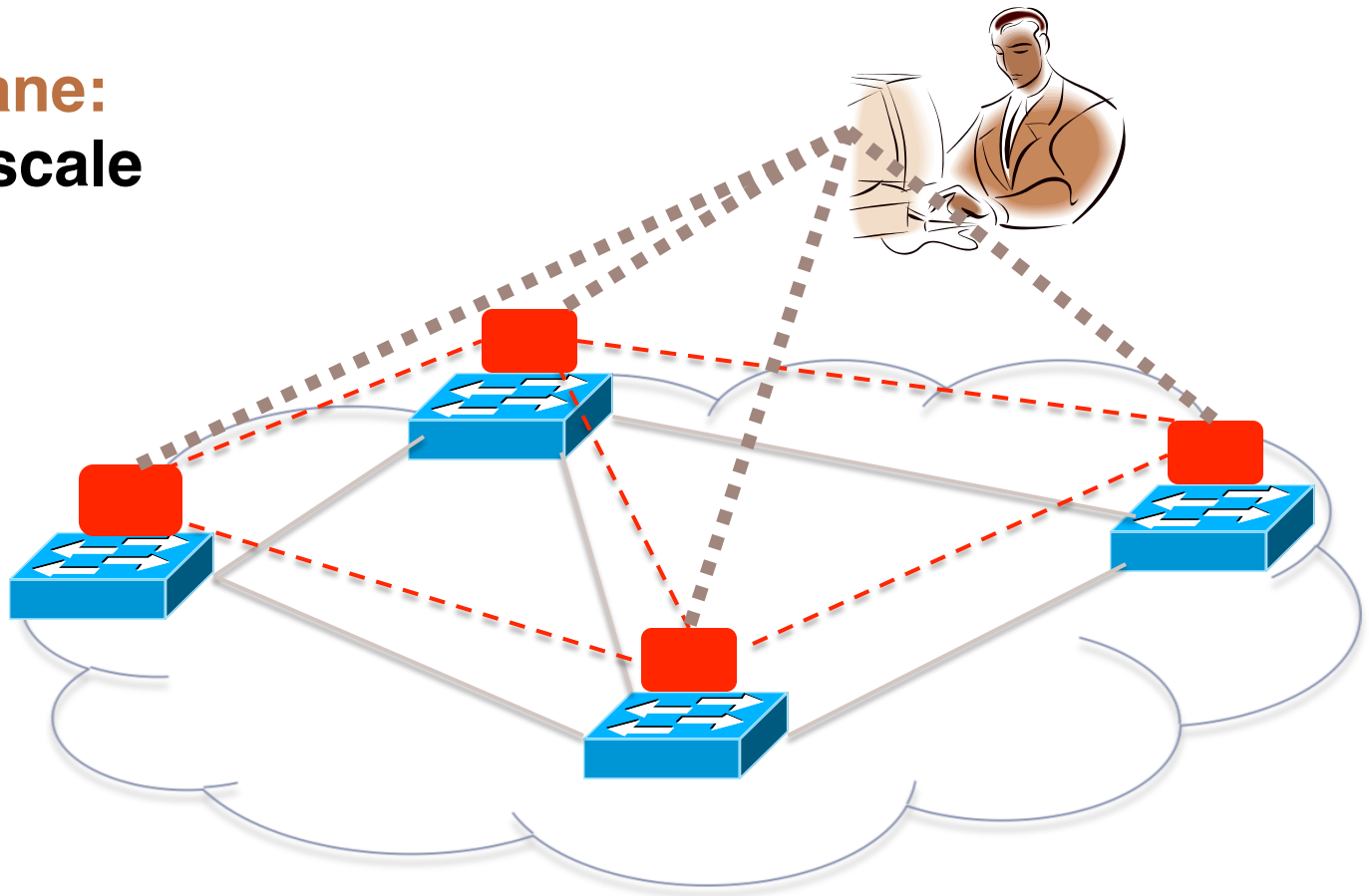


Track topology changes, compute routes, install forwarding rules

Traditional Computer Networks

Management plane:

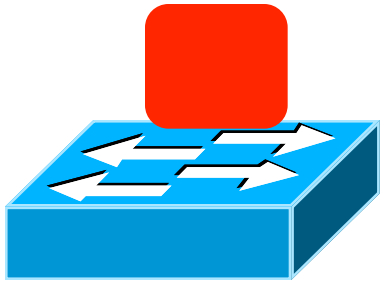
Human time scale



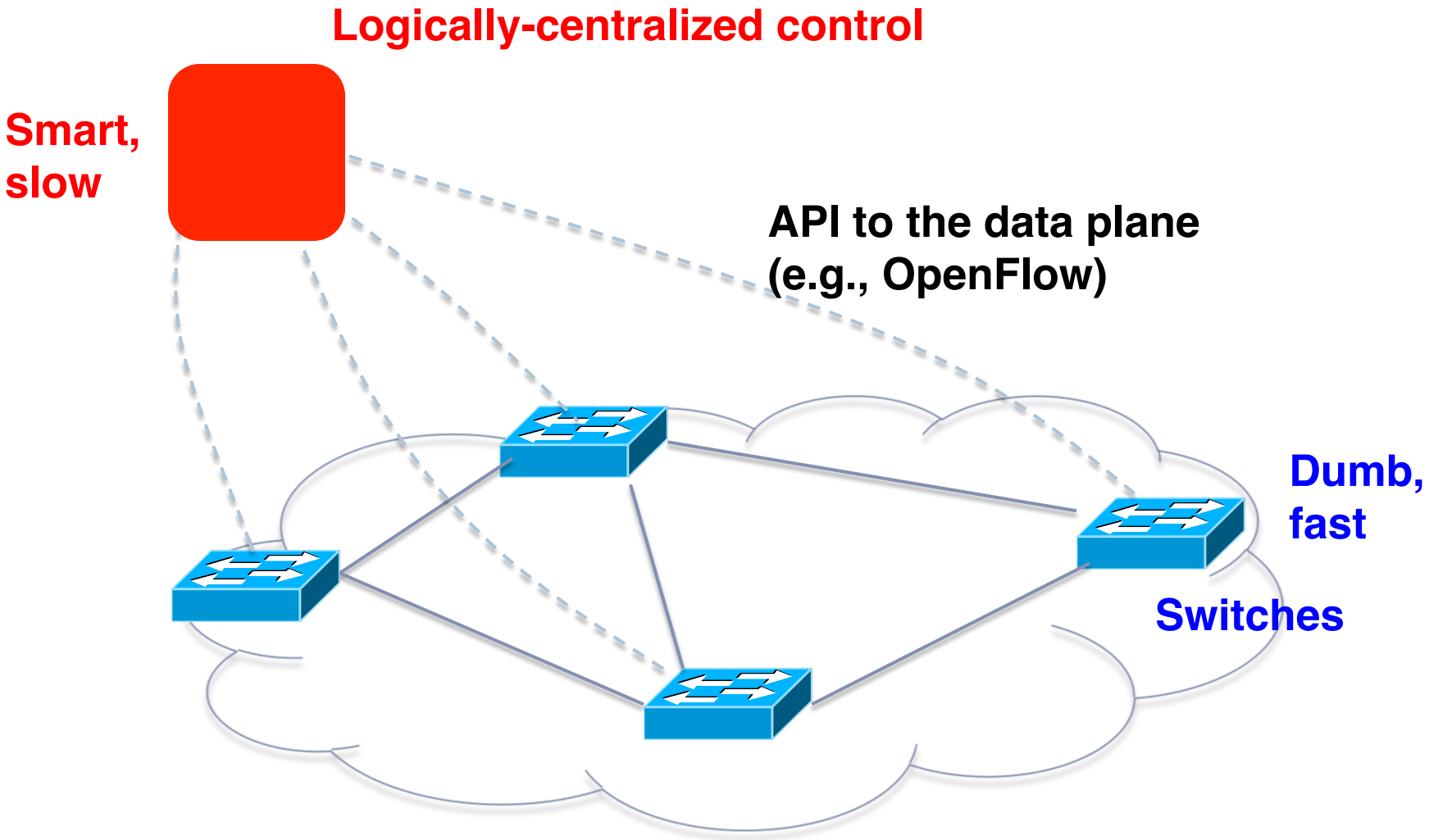
Collect measurements and configure the equipment

Death to the Control Plane!

- Simpler management
  - No need to "invert" control-plane operations
- Faster pace of innovation
  - Less dependence on vendors and standards
- Easier interoperability
  - Compatibility only in "wire" protocols
- Simpler, cheaper equipment
  - Minimal software



Software Defined Networking (SDN)



Data-Plane: Simple Packet Handling

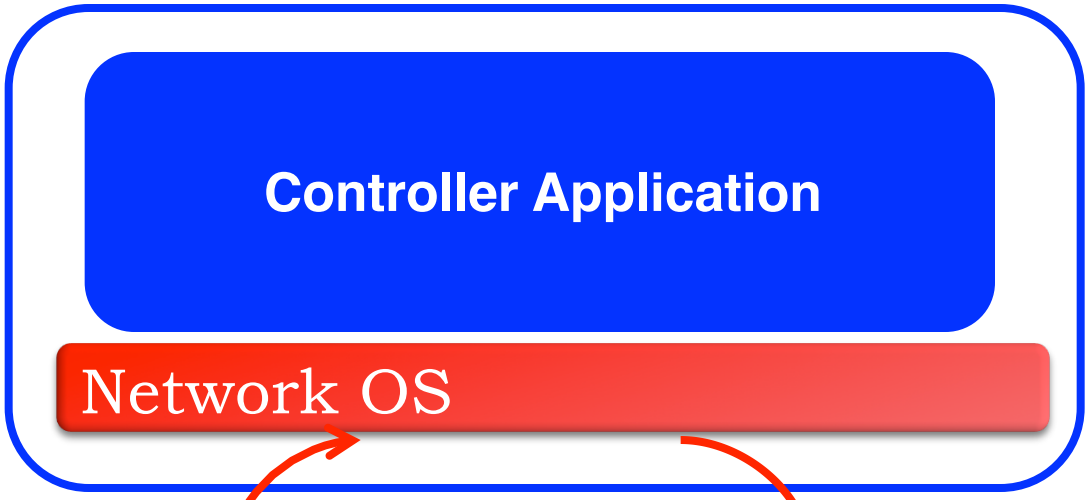
- Simple packet-handling rules
  - Pattern: match packet header bits
  - Actions: drop, forward, modify, send to controller
  - Priority: disambiguate overlapping patterns
  - Counters: #bytes and #packets

1. src=1.2.*.*, dest=3.4.5.* → drop
2. src = *.*.*.*, dest=3.4.*.* → forward(2)
3. src=10.1.2.3, dest=*.*.*.* → send to controller

Unifies Different Kinds of Boxes

- Router
  - Match: longest destination IP prefix
  - Action: forward out a link
- Switch
  - Match: destination MAC address
  - Action: forward or flood
- Firewall
  - Match: IP addresses and TCP/UDP port numbers
  - Action: permit or deny
- NAT
  - Match: IP address and port
  - Action: rewrite address and port

Controller: Programmability



Events from switches
Topology changes,
Traffic statistics,
Arriving packets

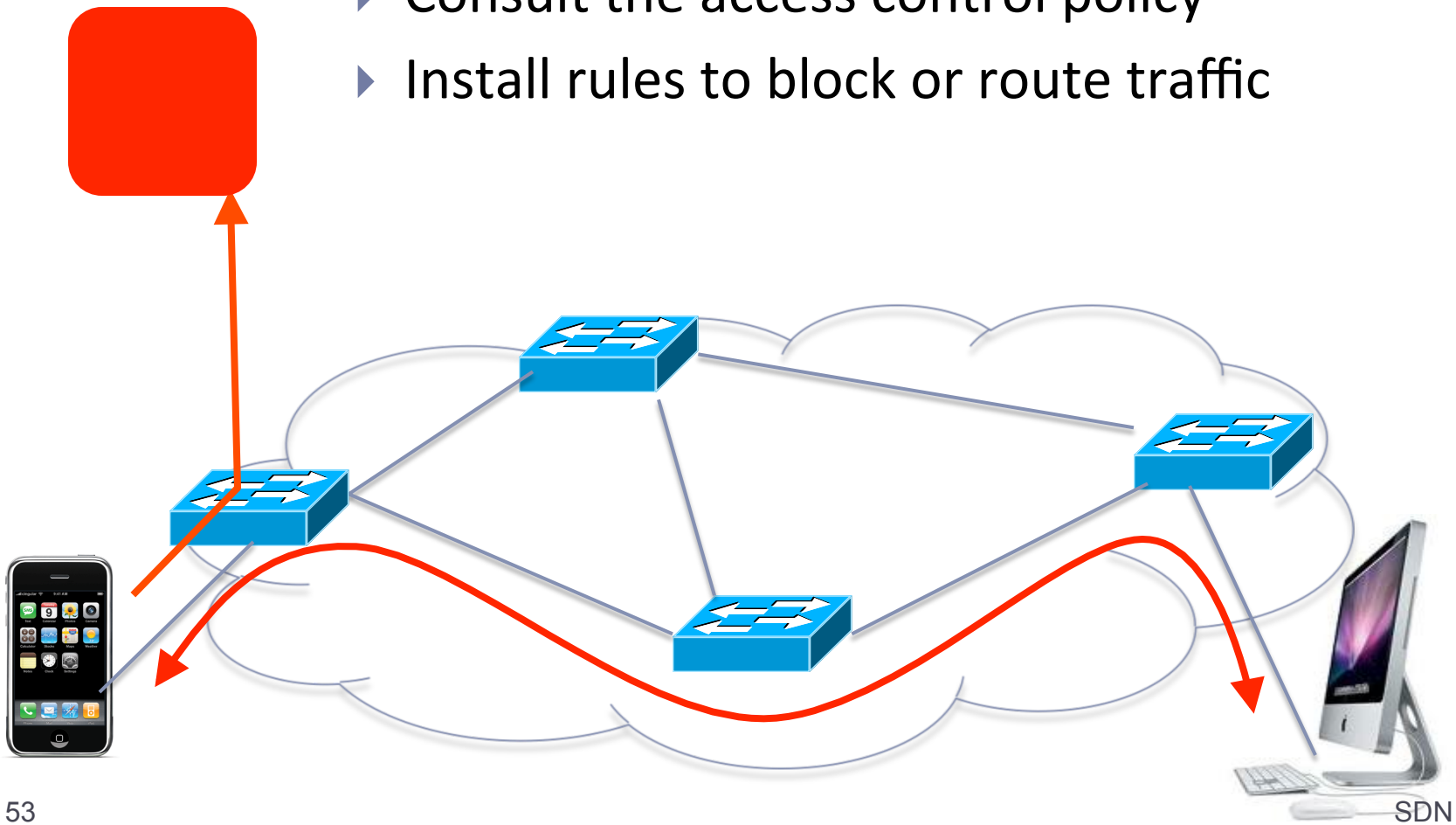
Commands to switches
(Un)install rules,
Query statistics,
Send packets

Example OpenFlow Applications

- **Dynamic access control**
- **Seamless mobility/migration**
- **Server load balancing**
- **Network virtualization**
- Using multiple wireless access points
- Energy-efficient networking
- Adaptive traffic monitoring
- Denial-of-Service attack detection

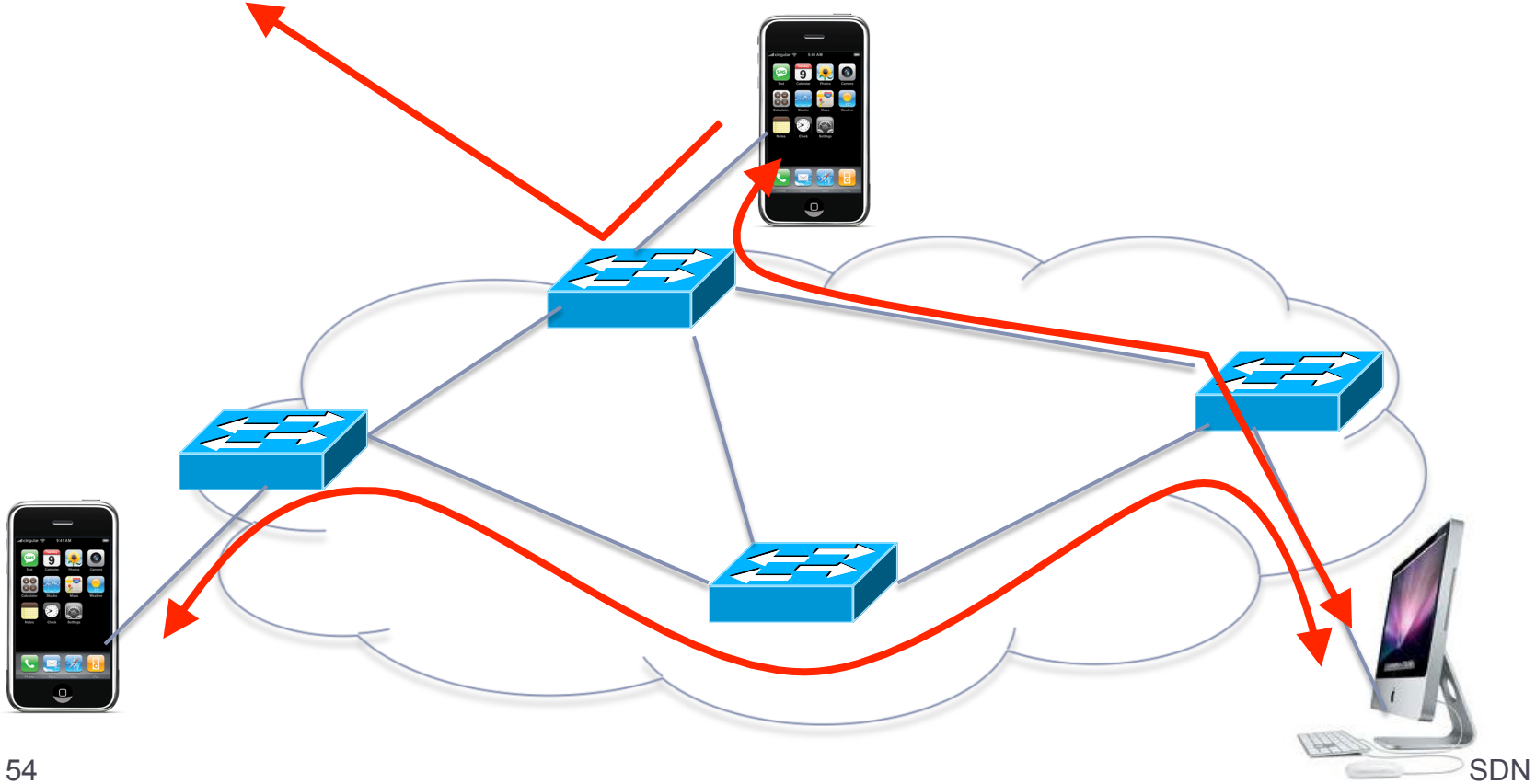
E.g.: Dynamic Access Control

- Inspect first packet of a connection
- Consult the access control policy
- Install rules to block or route traffic



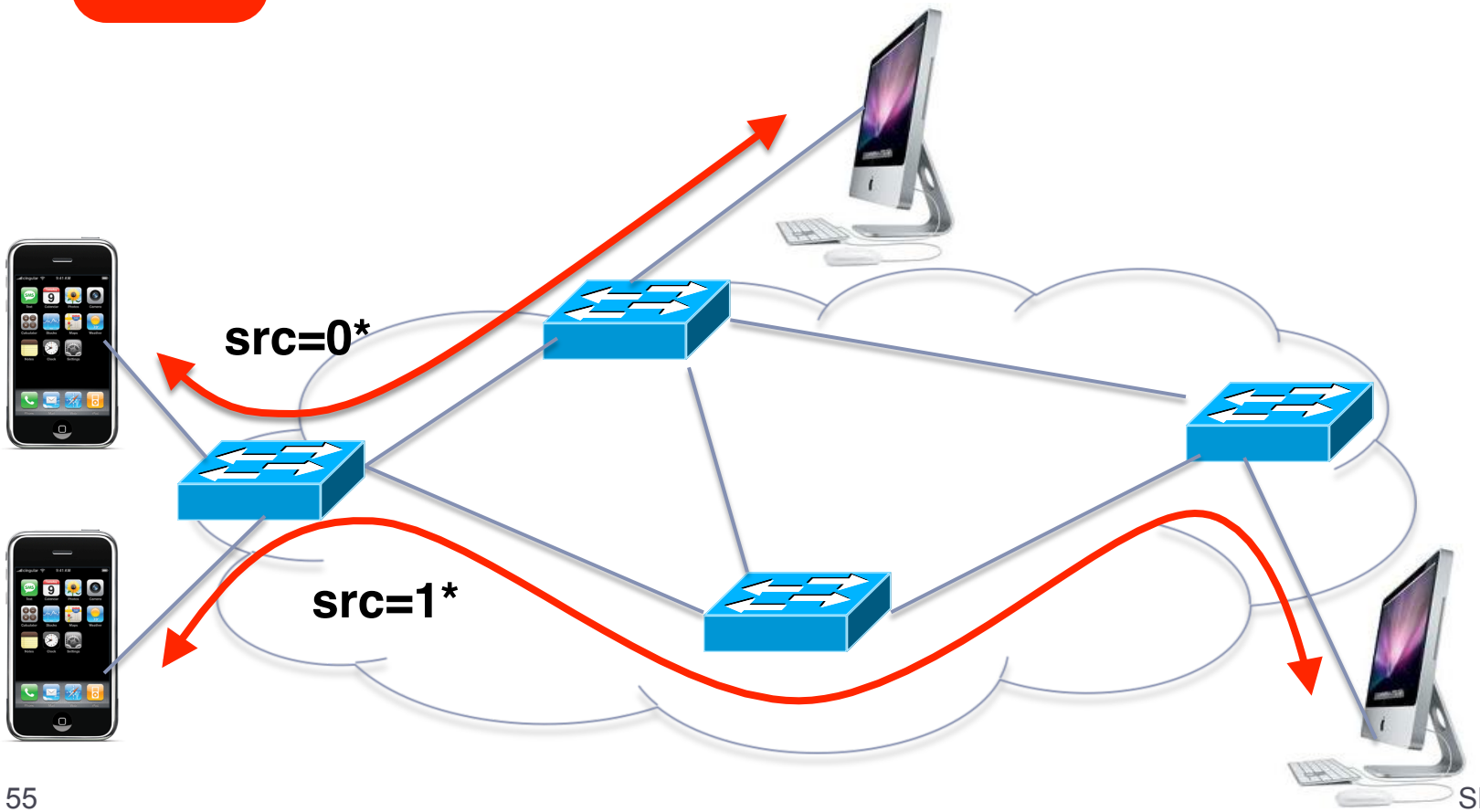
E.g.: Seamless Mobility/Migration

- See host send traffic at new location
- Modify rules to reroute the traffic



E.g.: Server Load Balancing

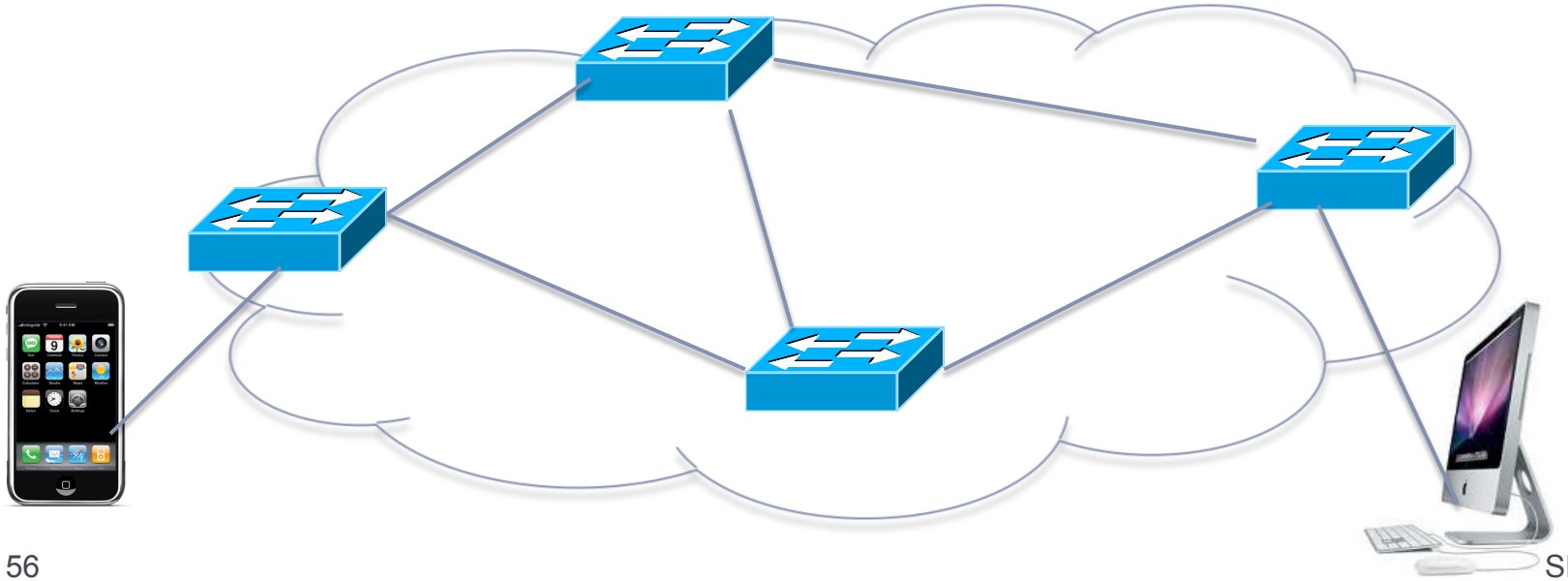
- Pre-install load-balancing policy
- Split traffic based on source IP



E.g.: Network Virtualization

Controller #1 **Controller #2** **Controller #3**

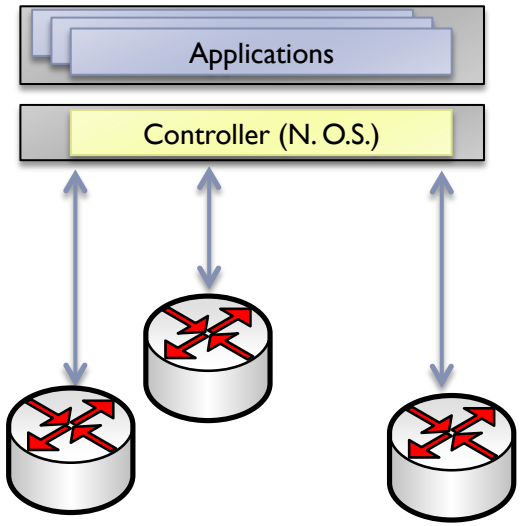
Partition the space of packet headers



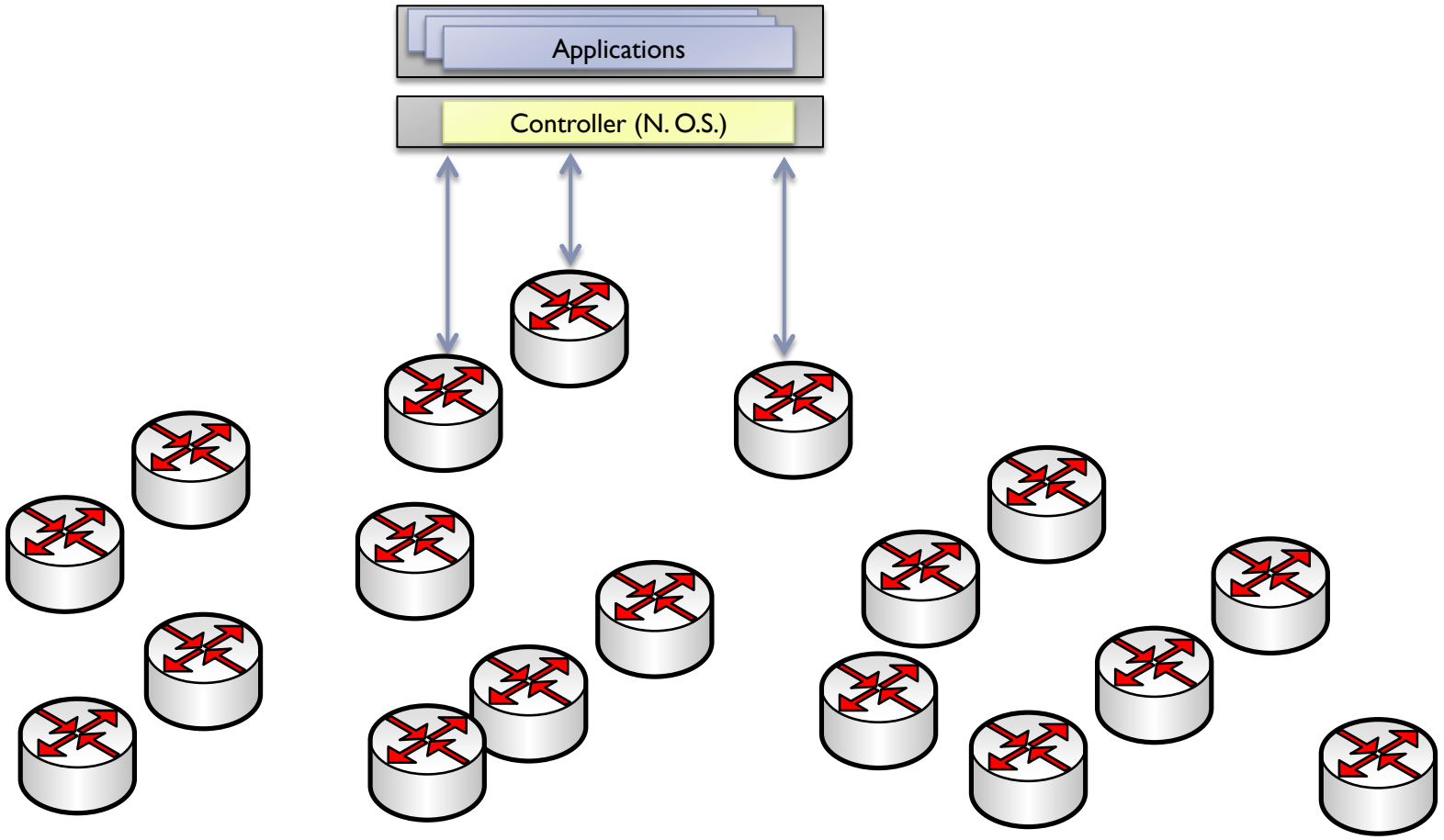
OpenFlow in the Wild

- Open Networking Foundation
  - Google, Facebook, Microsoft, Yahoo, Verizon, Deutsche Telekom, and many other companies
- Commercial OpenFlow switches
  - HP, NEC, Quanta, Dell, IBM, Juniper, …
- Network operating systems
  - NOX, Beacon, Floodlight, Nettle, ONIX, POX, Frenetic
- Network deployments
  - Eight campuses, and two research backbone networks
  - Commercial deployments (e.g., Google backbone)

Controller Availability



Controller Availability

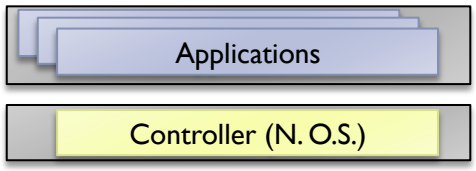


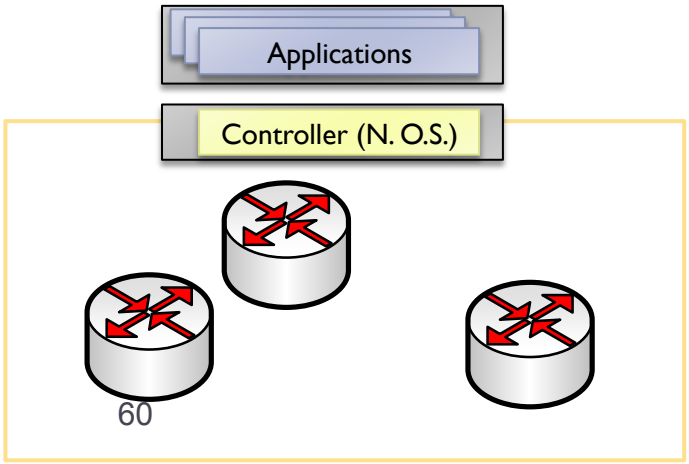
Controller Availability

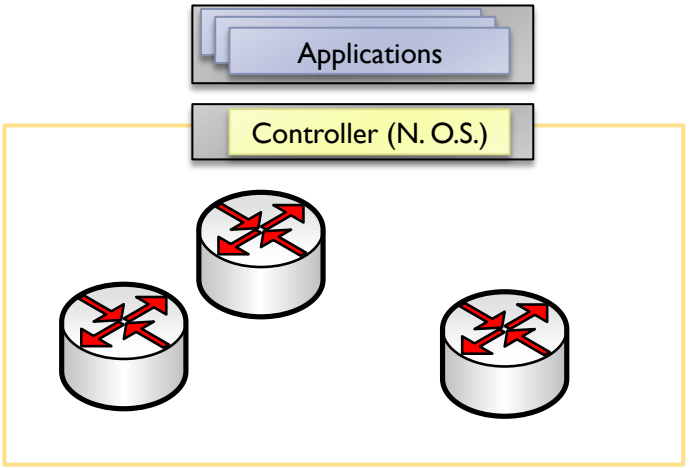
"control a large force like a small force: divide and conquer"

--Sun Tzu, Art of war

- How many controllers?
- How do you assign switches to controllers?
- More importantly: which assignment reduces processing time
- How to ensure consistency between controllers

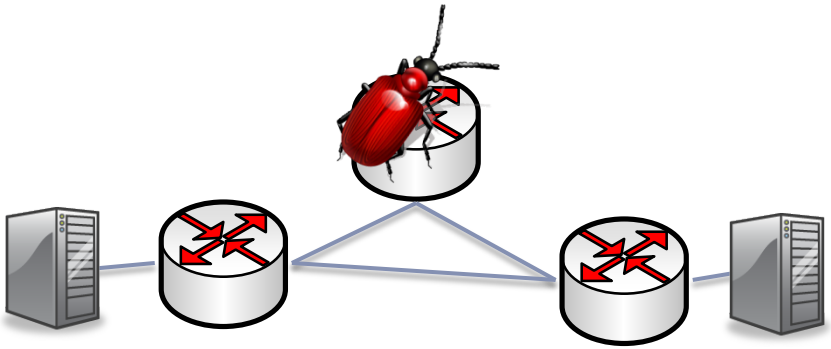






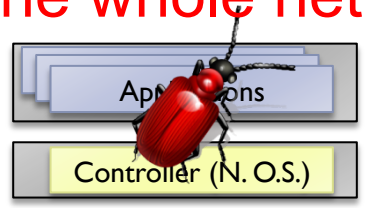
SDN Reliability/Fault Tolerance

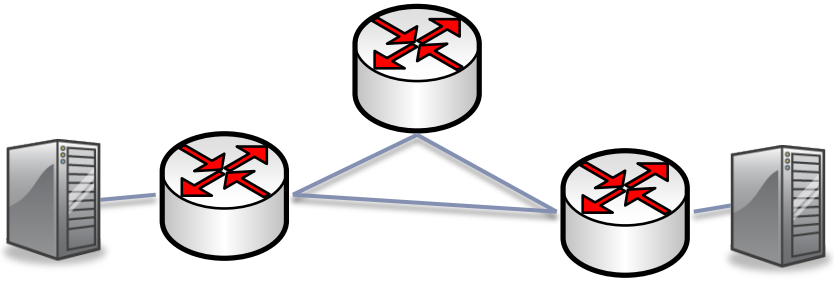
Existing network survives failures or bugs in code for any one devices



Controller: Single point of control

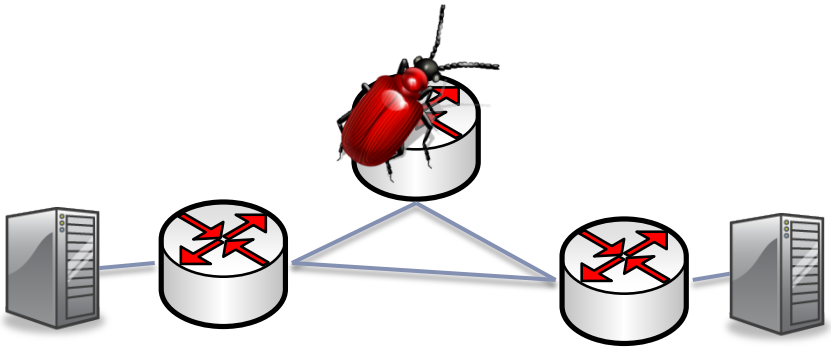
- Bug in controller takes the whole network down





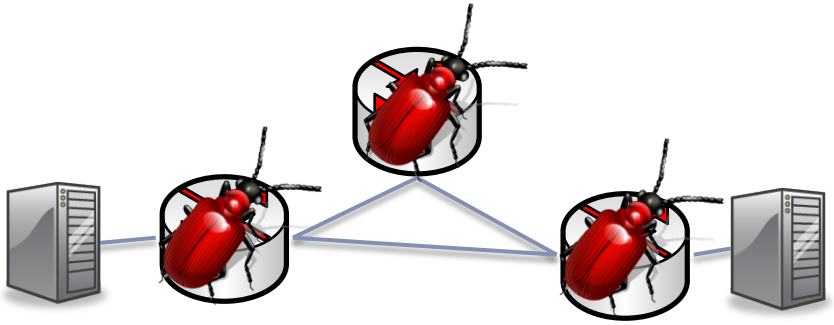
SDN Reliability/Fault Tolerance

Existing network survives failures or bugs in code for any one devices



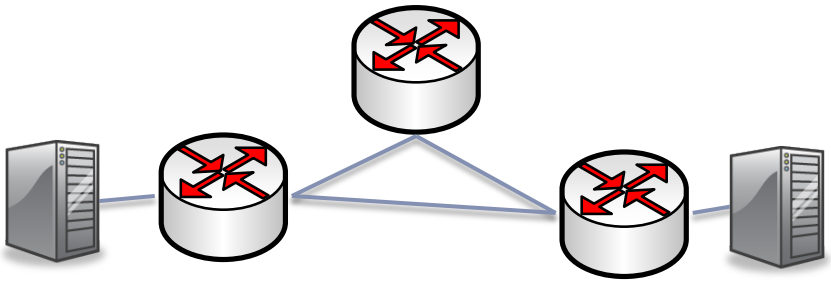
Controller: Single point of control

- Bug in controller takes the whole network down
- Single point of failure



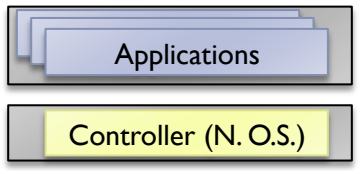
SDN Security

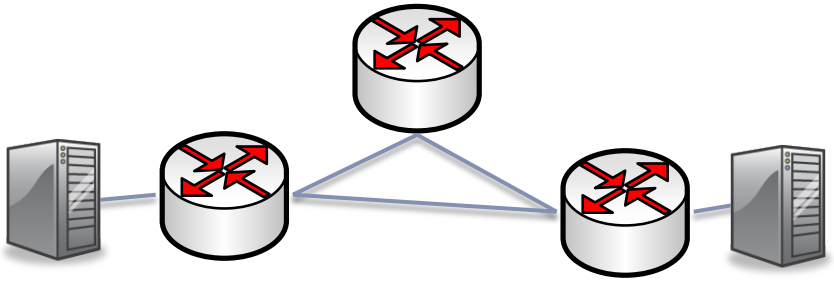
If one device in the current networks are compromised the network may still be safe



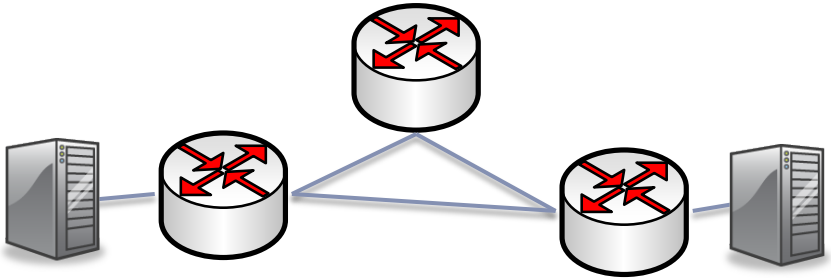
Controller: Single point of control

- Compromise controller





SDN Security



Controller: Single point of control

- Compromise controller
- Denial of Service attack the control channel