Data Center Networks

Introduction

1. Cover the following topics:
    - Metrics for network performance evaluation
    - Tools for testing and debugging data center networks
    - Case studies of analyzing data center network traffic
2. Answer the following questions:
    - What can go wrong in data center networks?
    - What are the tools for testing and debugging?
    - What are the tools for measurements and performance analysis?
    - Case studies of measurements of data center networks?

Data Center Networking Overview

1. Metrics
    - Latency: Time between two points
    - Throughput: Events per unit time
    - Utilization: How well the network infrastructure is used
    - Scalability: As you increase the offered load into the system, does the performance experienced by the applications remain unchanged?
        - Is my performance affected by other applications running?
2. Performance Evaluation
    - Modeling: Mathematical way of representing what's going on in the system
        - Computer systems are mostly heuristics, can be difficult
        - More approximate than a simulation
    - Simulation: Taking the real system and writing a program that simulates it. Analyze the performance of the simulator
        - Can drive simulation with traces from a real system
    - Implementation and Measurements: Build the system and study its performance
        - Expensive
    - Ideally, we first model, then simulate, then implement

What Can Go Wrong?

1. Typical networking related problems
    - Forwarding loops
    - Link failures
    - Forwarding inconsistencies -> often leads to forwarding loops
    - Unreachable hosts
    - Unwarranted access to hosts (which should not be reachable)
2. Data Center vs Traditional Networks
    - Forwarding loops
        - In traditional networks caused by failure of spanning tree protocols
    - Link failures
        - Response is different, but problem is the same
    - Unreachable hosts
        - In traditional networks due to errors in ACLs or routing entries
        - In SDNs due to missing forwarding entries
    - Unwarranted access to hosts
        - In traditional networks due to errors in ACLs
        - In SDNs caused by unintended rule overlap
3. Challenges for Data Center Networks
    - Potential control loops

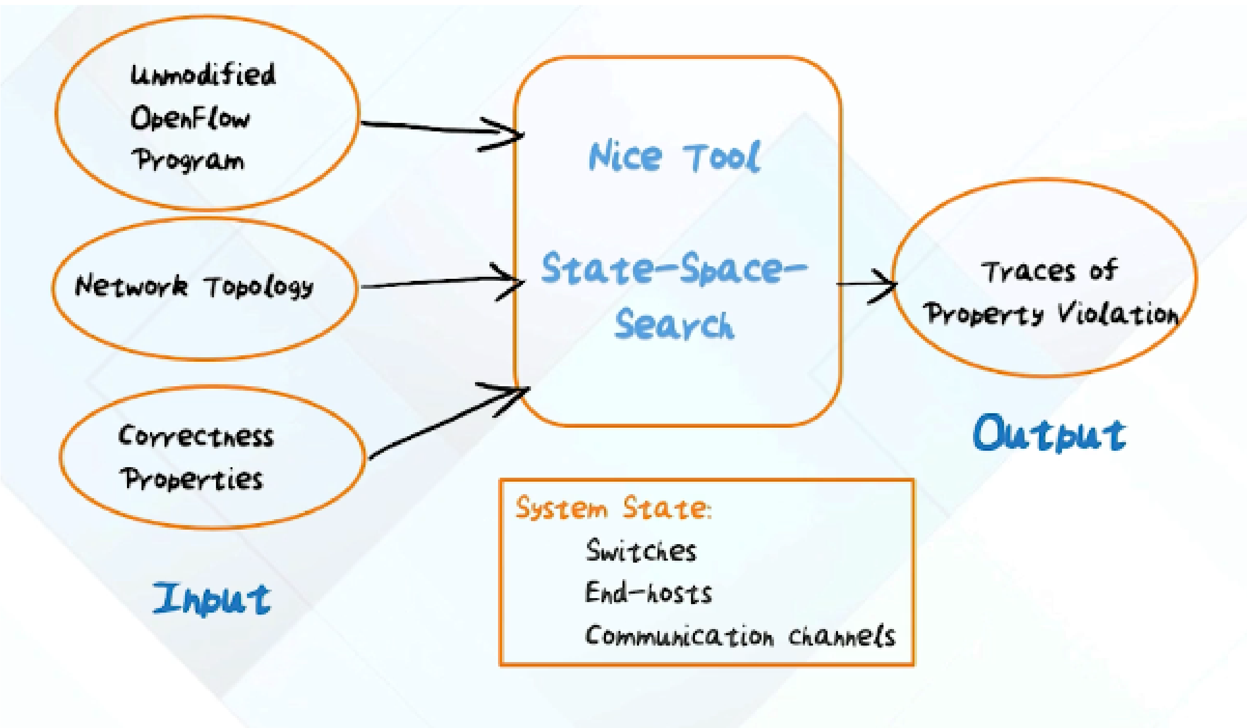
    - Switches, controller, application
  - End-to-end behavior of networks
    - Networks are getting larger
    - Network functionality becoming more complex
    - Partitioning functionality: Unclear boundary of functionality between network devices and external controllers
4. Effect of Network Bugs
  - Unauthorized entry of packets into a secured zone
  - Vulnerability of services and the infrastructure to attacks
  - Denial of critical services
  - Effect network performance and violation of SLAs

Tools for Testing, Debugging, Verification

1. Approaches to Testing and Debugging
  - Domain-specific languages to minimize errors
    - Don't use C to program controller to limit kinds of errors
  - Limited set of principles
  - Symbolic execution and model checking
    - Constrain state space to check the model
  - Static analysis of the network state
    - Run network, observe state, do a post-mortem later on
  - Live debugging
    - As the applications are running, use a network debugger to probe

N.I.C.E

1. Can we build a tool for systematically testing data center applications?
  - Data plane
    - Huge space of possible packets
  - Control plane
    - Huge space of event ordering on the network
  - Light at the end of the tunnel
    - Equivalence classes of data packets
    - Application knowledge of event ordering
2. N.I.C.E: No bugs In Controller Execution
  - Tool for automatically testing OpenFlow applications
    - Goal: Systematically test possible behaviors to detect bugs
  - Possible state-space is exponential
    - Combinatorial explosion with brute force approach
  - NICE's magic sauce
    - State-space exploration via Model Checking (MC)
    - Combine Symbolic Execution (SE) with Model Checking to prevent state-space explosion
  - Take advantage of knowledge of the application properties to reduce the state-space



N.I.C.E Diagram

OFRewind

1. Static analysis of OpenFlow programs
2. How does it work?
   - In production: Record state (events and traffic) with minimal overhead
   - Later: Replay state at convenient pace
   - Troubleshoot: Reproduce problems at chosen times/locations
3. Keys to scalability of this approach
   - Record control plane traffic
   - Skip/aggregate data plane traffic
   - Replay: Best effort as opposed to deterministic
     - Identify performance problems or correctness issues, so we don't necessarily need a deterministic execution
   - Over-arching goal: Partial recording/replay of chosen times/locations to reproduce problems
4. How to use OFRewind
   - Deploy OFRecord in production
     - "Always on" OF messages, control plane, data plane summaries
     - Selection rules as necessary
   - Deploy OFRecord in the lab
     - Localize bugs and validate bug fixes

Ndb

1. Tool for live debugging of errant network behavior (similar to gdb for program control flow)
2. Approach
   - Use SDN architecture to systematically track down network bugs
   - Capture and reconstruct the sequence of events leading to the errant behavior
   - Network breakpoints defined by the user

        - Filter (header, switch) to identify the errant behavior
    - Back trace generation
        - Path taken by the packet
        - State of the flow tables at each switch
3. How it works
    - Breakpoint is a filter on packet header
        - e.g. <switch S; IP_src = A; IP_dst = B; TCP_port = 22>
        - Switches send "postcard" on matching entries to a central collector
        - Collector stores the postcards to construct a "backtrace"
    - Collector can become a bottleneck
        - Instead of applying rule to all switches, apply to a subset to reduce total amount of collected traffic
        - Collector can be parallelized, don't have to be centralized

Header Space Analysis and Netplumber

1. Elements of Header Space Analysis
    - General model that is agnostic to the protocols and network topologies
    - Models the packet header (length L) as a point in an L-dimensional hyperspace
    - Models all network boxes as a transformer on the packet header space
    - Defines an algebra for this transformation
        - Composable, invertible
    - Models all kinds of forwarding functionalities regardless of specific protocols and implementations
2. Using the Model
    - All traffic flows can be expressed as a series of transformations using the algebra
    - Allows asking questions such as
        - Can two hosts communicate with each other?
        - Are the forwarding loops?
        - Are there network partitions?
3. Netplumber
    - A system built on header space analysis
    - Creates a dependency graph of all forwarding rules in the network and uses it to verify policy
        - Nodes in the graph -> Forwarding rules in the network
        - Directed edges -> Next hop dependency of the forwarding rules
4. Represent forwarding policy as a dependency graph
    - Flexible policy expression
        - Probe and source nodes are flexible to place and configure
    - Incremental update
        - Only have to trace through dependency sub-graph affected by an update to the forwarding policy
    - Parallelization
        - Can partition dependency graph into clusters to minimize inter-cluster dependences

Veriflow

1. Tackles the problem of network-wide verification of traffic flows
2. Goal: Detect routing loops, black holes, access control violations
3. Approach
    - Interpose verification layer between SDN controller and the network elements
    - Formulate network invariants from the SDN controller
    - Construct a model of network behavior
    - Monitor the network for violation of invariants

Network Traffic Characteristics

1. Data Center Network Traffic Study
   - Are links over-subscribed?
   - Is there sufficient bisection bandwidth?
   - Is centralization (via SDN controller) feasible?
2. Setup for the Study
   - Classic model of data center network
     - Core (L3), aggregation (L2), edge (Top-of-Rack-L2) layers
   - 10 data centers from three classes
     - University, private enterprise, cloud
   - User community
     - Internal (university, private) and external (cloud)
   - Methodology
     - Analyze running applications using packet traces
     - Quantify network traffic from applications
3. Results Summary
   - Significant amount of small packets (~50% less than 200 bytes)
     - TCP acks, "I am alive" messages
   - Importance of connection persistence
   - Traffic distribution
     - Clouds: Most traffic (75%) within a rack -> good colocation of application components
     - Other DCs: 50% inter-rack -> un-optimized placement
   - Link utilization
     - Core > Aggregation > Edge
     - Bisection bandwidth sufficient (only 30% of the bisection used)
4. Insights from the Study
   - Are links over-subscribed? No
     - 75% traffic within a rack
     - Core links utilization < 25%
     - Need better load balancing, VM placement, and VM migration
   - Is there sufficient bisection bandwidth? Yes
     - Small packet sizes
     - Utilization < 30%
   - Is centralization feasible? Yes
     - Most apps use TCP/IP so setting up switches and amortizing the cost is worth it

Classification of Traffic

1. Types of traffic
   - D2C Traffic: Traffic exchanged between Yahoo servers and clients
   - D2D Traffic: Traffic exchanged between different Yahoo servers at different locations
   - Client: Non-Yahoo host connect to Yahoo server
2. Methodology for collecting traffic data
   - Anonymized NetFlow datasets collected at the border routers of five major Yahoo data centers
     - Dallas (DAX), Washington DC (DCP), Palo Alto (PAO), Hong Kong (HK), United Kingdom (UK)
   - Meta data collected
     - Timestamp, source and destination IP address, transport layer port number, source and destination interface on the router, IP protocol, number of bytes and packets exchanged
3. Key findings of the study
   - Yahoo data centers are heirarchically structured
   - D2D traffic patterns
     - D2C triggered traffic: Smaller with higher variance commensurate with user dynamics
     - Background traffic: Dominant and not much variance

- Highly correlated traffic at data centers
  - Replicated services at different data centers
  - Implications for distributing services at multiple data centers

Structure of Google WAN

1. Google WAN
   - Characteristics
     - Global user base
     - QoS needs: High availability and quick response
   - Implications
     - Rapid movement of large data across WAN
   - Organization of the network
     - I-scale: Internet facing
     - G-scale: Inter-data center
2. G-Scale Network
   - OpenFlow powered SDN
   - Proprietary switches from merchant silicon and open source routing stacks with OpenFlow support
   - Each site
     - Multiple switch chassis
     - Scalability (multiple terabits of bandwidth)
   - Fault tolerance
     - Sites inter-connected by G-scale network
     - Multiple OpenFlow controllers
     - No single point of failure

Conclusion

1. Tools and techniques for testing, debugging, verification, and performance analysis of data center networks