Data Center and Cloud Cheat Sheet

by Francisco_1088 via cheatography.com/50986/cs/13991/

TCAM

Compares data against predefined ruleset in one operation
Return action or address with first match
Rules consist of 1, 0, X's
OF Table entries contain Match, Action, Counter, Prio, Timeout

Geometric Representation of Rules

Rules can be specified by prefix/length pairs or operator/number (range)
Rule with d fields -> d-dimensional hyper-rectangle
Match condition is finding highest priority hyper-rectangle enclosing P
When rectangles overlap, smallest rectangle "wins"

Edge Network Mgmt

Mgmt is 80% of IT budget and responsible for 62% of otuages
Networks should be truly transparent
Challenges:
-Large network scalability
-Flexible policies: custom routing, measurement and diagnosis, access control
-Commodity switches: small memory, expensive and power hungry, more link speed, storing lots of states, monitoring flows, qos
DC Networks:
-VM migration, load balancing, task scheduling, anomaly detection/isolation

Difane

DIFANE Design goals:
-Scale with network growth
-Improve per-packet performance: always keep in dataplane
-Minimal switch modification: no change to dataplane hardware
Difane stages:
-Controller proactively generates rules and distributes to authority switches
-Controller proactively partitions rulespace in wildcards and distributes to all switches
-Ingress switches receive unknown flows and they contact Authority switch in correct wildcard space

DIFANE 2

-Authority switch forwards packet to correct destination and caches corresponding rule in ingress switch for future packets
Caching wildcard rules:
-Controller creates new rules for lower priority rules that overlap with high priority rules (e.g. R1 0-7, 5-6, R3 6-7,0-15, they overlap in 6-7,5-6, so controller creates R3 rule for 6-7,0-3 and 6-7,7-15)
-Rules must be correctly partitioned by controller to ensure optimal usage of TCAM, some cuts are better than others

DIFANE 3

Network dynamics:
-Policy change at controller: Timeout cache rules, Change authority rules, No change for partitions
-Topology change at switch: No change in cache rules, no change in authority rules, Change in Partition rules
-Host mobility: Timeout cache rules, No change in authority or partition rules

Caching in Buckets

Partition rulespace in a grid of buckets
-Larger buckets mean more rules are cached each time
-Smaller buckets means more buckets need to be cached
-Partition until number of associated rules is bounded
-Sweetspot for bucket size is in a region, smaller and larger than this leads to memory overflow
-CAB reduces control network BW, flow setup latency and controller load
-Fully compatible with OF standard, resolves dependencies wildcard rule caching

Cloud Security

Typical practices:
-Reinforce application security, strong network perimeter security
-Access control inside cloud for app/service/tenant isolation
-Gauge risk control when using public cloud
Problems:
-Placing new security hardware is not easy
-Security devices are typically shared, misconfiguration in one compromises many services, apps and hosts
-Tight work between network and security teams, high cost and low efficiency

Policy Aware Switch

-Makes forwarding decisions based on various factors, such as previous hop, input port, source/dest address

Cloud NaaS

Features:
-Virtual network isolation
-Custom addressing
-Service differentiation
-Flexible middlebox inteprosition
Cloud controller: provides VM instance management, self-service provisioning, host virtual switch interconection
Network controller: provides VM placement directives to cloud controller, generates virtual network between VMs, Configures physical and virtual switches

Hybrid Security Architecture

-Tenants everywhere -> Middlebox anywhere
-Flexible traversal: traffic-specific, middlebox type, arbitrary number and order
-Decouple networking from security, creating appliance layer
App layer: App VMs with security groups
Appliance layer: Traversal path of middleboxes
Network layer: Only cares about packet delivery
-Forwarding: MAC rewrite for L2, IP in IP for L3

HSA Benefits

-Scalable and flexible provisioning
-Facilitates virtualization, simplifies service development, testing, deployment and troubleshooting
-Enables dynamic and heterogeneous service provisioning
-Minimize misconfiguration impacts

SDN Security

Bottlenecks: Weak OF Agent CPU, limited message processing capabilities, Limited TCAM/SRAM resources->table overflows
Solution: Leverage NFV to build a software-based defense line
NFV in edge clouds:
-Elastic resource allocation
-Network function as a service
-Rapid innovation

SDN Shield

-Controller monitors switch packet-in message rate from each switch
-When one switch-s rate approaches saturation: countermeasure
-Use a second Attack Mitigation Unit
How to Identify Legitimate Flows:
-Use statistical filtering
Conditional Legitimate Probability:
-Analyze header field distribution
-Compare most recent measurement to reference profile
-Build scoreboard to calculate new flow's legitimacy probability
-Threshold to control the rate of passed flows

PacketScore

1: Detect Attack -> monitor key parameters of traffic destined to protected targets, contain by limiting resource consumption
2: Differentiate attacking packets from legitimate ones in suspicious traffic: compare against baseline and use CLP to compute likelihood of each suspicious packet of being legitimate
3: Discard suspicious packets selectively comparing CLP with dynamic threshold

Attack types

Endpoint: overload a victim or stub network -> Easily isolated by upstream routers, attacking packets have victim IP/subnet
-Monitor traffic rate, flow rate towards each host/stub -> large number of targets monitored
-Use Bloom-filter to catch targets under attack, use DDoS control server to aggregate and correlate
Infrastructure: Overload some choke-point (router uplink) -> hard to isolate unless packet traceback infrastructure is in place
-Monitor traffic parameters on links in routers

CLP 1

$$CLP(p) = \frac{N_n \cdot JP_n(A=a_p, B=b_p, C=c_p, \ldots)}{N_n \cdot JP_n(A=a_p, B=b_p, C=c_p, \ldots) + N_a \cdot JP_a(A=a_p, B=b_p, C=c_p, \ldots)}$$

$$= \frac{N_n \cdot JP_n(A=a_p, B=b_p, C=c_p, \ldots)}{N_m \cdot JP_m(A=a_p, B=b_p, C=c_p, \ldots)}$$

$$= \frac{\rho_n}{\rho_m} \cdot \frac{JP_n(A=a_p, B=b_p, C=c_p, \ldots)}{JP_m(A=a_p, B=b_p, C=c_p, \ldots)} \ldots\ldots\ldots\ldots Eq.(1)$$

N_n = total number of legitimate, i.e. normal, packets over a certain observation interval ;
N_a = total number of attack packets over a certain observation interval ;
N_m = total number of packets over a certain observation interval $= N_n + N_a$;
ρ_m = current measured utilization of the system ;
ρ_n = nominal/baselined utilization of the system ;

If packet attributes are independent, Joint Probability Mass function can be separated in P(A=a)*P(B=b)...

VXLAN

-Network virtualization technology to improve scalability problems in large cloud deployments
-VLAN-like encapsulation, encapsulates L2 frames in UDP packets with port 4789 using a VNI
-Endpoints are called VTEPs, and may be virtual switches, hypervisors or NVGREs
-Overlay network is usually a multicast cloud
-NVGRE uses GRE to encapsulate L2 frames in L3 packets across L3 networks

ARISTA VXLAN

Challenges: Oversubscription, Scalability, Cost, Mobility and Latency
Network virtualization: Create overlay networks on top of physical network infrastructure
VXLAN 24 bit ID -> 16M networks
-Can cross L3, 50bytes of overhead
-VMs don't see tag
-L2 broadcast is replaced by IP multicast
Benefits:
-VLAN sprawl
-Single fault domains
-Scalability beyond 4096 segments
-Non-proprietary fabric
-IP mobility
-Physical cluster size and locality improves
-Better multitenancy

Arista VXLAN 2

VTEP: Tunnel endpoint
VXLAN GW: Bridges VXLAN to non-VXLAN environment (HW or SW)
VNI: Identifies VXLANs
VTI: Terminates a VTEP
VXLAN Segment: L2 overlay network over which VMs communicate, only VMs within same VXLAN segment can communicate
OVSDB: Allows management of Open vSwitches, create or delete ports, tunnels, and queues

VXLAN BGP-EVPN

VXLAN Overlay is an L2 broadcast domain identified by a VNI
VXLAN encap:
-Outer header -> IP source and dest from VTEP endpoints, L2 source from VTEP source, L2 dest from next L3 hop, UDP port dest 4789
Gateway types:
L2-> VLAN to VXLAN bridging
L3-> VXLAN to VXLAN routing

VXLAN Flood and Learn

VNI is mapped to a multicast group on a VTEP
Broadcast, Unknown Unicast and Multicast traffic is flooded to the multicast group of the VNI
Remote VTEPs of the group learn host MAC, VNI and source VTEP IP from flooded multicast traffic
Unicast packets for the host are sent directly to the source VTEP IP
Encapsulated packet:
UDPd: 4789, IPd: remote VTEP/multicast group, IPs: source VTEP, Md: remote VTEP/multicast MAC, IPd: Remhost, IPs: Sourcehost, Md: Remhost/Broadcast, Ms: Sourcehost

VXLAN EVPN

1 L3 VNI per VRF per VTEP
1 L2 VNI per L2 segment, multiple L2 VNIs per tenant
BGP minimizes network flooding and allows VTEP peer discovery and authentication
All VTEPs keep the same IP address for L2 VNIs
Process:
-Host sends out GARP when they come online
-Local VTEP creates local ARP cache and advertises through BGP as Route Type 2
-Remote VTEP puts IP-MAC info into remote ARP cache and suppresses ARP for this IP
-VTEP floods if no match is found in cache

VXLAN BGP EVPN

Asymmetric IRB: different path from source to dest and back, VTEP must be configured with both source and dest VNIs for both l2 and l3
Symmetric IRB: same path to destination and back, ingress VTEP routes from source VNI to L3 VNI and changes inner dest MAC to egress VTEP router MAC
Route Types:
Type 2: MAC advetisement -> L2 VNI MAC/MAC-IP -> MAC and ARP resolution
Type 5: IP Prefix Route -> L3 VNI route -> advertise prefix

MP BGP VXLAN

L2 traffic cannot traverse VNI boundaries
L3 traffic from one VRF is mapped to a L3 VNI
L3 traffic from different VRFs cannot traverse L3 VNI boundaries
BGP update sends Host MAC, Host IP, L3 VNI and VTEP
Remote VTEPs take Host MAC and put it in MAC table, and Host IP and put it in VRF (L3 VNI) IP table
Local host information is learned through conventional L2 learning and GARP, or through mgmt plane integration between VTEP and hosts

VXLAN with MPBGP

-Improves scalability
-Enables control plane learning of L2 end host and L3 reachability
-Reduced network flooding
-Optimal east-west and north-south forwarding
-VTEP discovery and authentication

IRB

Asymmetric	Symmetric
-Ingress VTEP does L2 and L3 lookup, egress VTEP only L2 lookup	-Both VTEPs perform L2 and L3 lookup
-All VTEP need all VNIs	-InterVXLAN traffic is encapsulated in L3 VNI, which identifies VRFs
-Ingress VTEP routes from source VNI to dest VNI	-ingress VTEP does not need to know dest VNI
-Not scalable	-Scalable

TCP in the DC

Not good for DC
-Adds latency
-Wastes buffer space
-Performs bad with shallow-buffer switches
DC Workloads:
-Partition/Aggregate (Delay sensitive, bursty)
-Short messages (delay-sensitive)
-Large flows (throughput sensitive)
Incast: Synchronized congestion from partition-aggregate workloads
-Seemingly underutilized links become overutilized in short burst causing unseen drops

DC Transport Requirements

-High burst tolerance
-Low latency
-High throughput
Traditional TCP:
-Window flow control: lost packets detected by missing ACKs
-W=BW x RTT -> awnd (receiver), cwnd (network), W = min(awnd,cwnd)
Algorithms to calculate cwnd: Tahoe, Reno, NewReno, DCTCP

TCP Tahoe and Reno

Tahoe	Reno
-3 DUP ACKS -> Fast Retransmit, set ssthresh to cwnd/2, reduce cwnd to 1 MSS, reset to slow start	-3 DUP ACKS -> Fast Retransmit and skip slow start, set cwnd to cwnd/2, enter fast recovery
-ACK time out (RTO) -> Slow start, cwnd -> 1MSS	-ACK time out (RTO) -> Slow start, cwnd -> 1MSS
	Fast recovery: wait for ACK for entire window before returning to CA, if no ACK enter slowstart

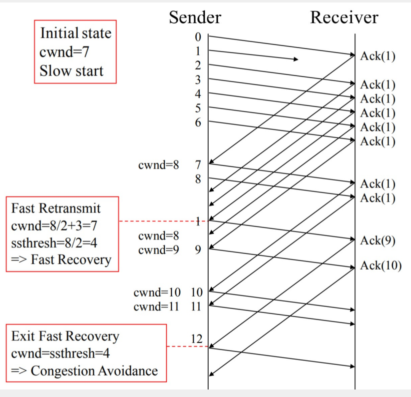
Data Center and Cloud Cheat Sheet

by Francisco_1088 via cheatography.com/50986/cs/13991/

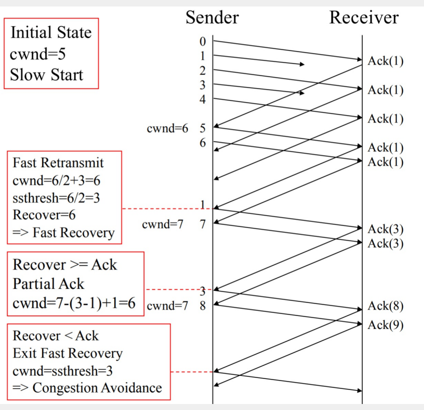
TCP

Slowstart: Start with cwnd =1, each ACK cwnd <- cwnd + 1, each RTT cwnd <- 2xcwnd (exponential)
CA: enter when cwnd >= ssthresh, each ACK cwnd<-cwnd+1/cwnd
-Each RTT: cwnd <- cwnd + 1
Fast Retransmit: flightsize = min(awnd,cwnd), sshthresh = max(flightsize/2,2)
-Enter slowstart cwnd=1

TCP Reno



TCP NewReno



New Reno

Remember last segment sent before Fast Retransmit
-Deal with partial ACK (new ACK does not cover last remembered segment, i.e. more packets lost before entering FR)
-Retransmit new lost packet too and remain in Fast Recovery, exit when ACK that covers last segment sent before FR is received)
-
sshthresh=max(flightsize/2,2*mss)
-cwnd=sshthresh+3*mss
-each new dupack cwnd=cwnd+mss
-when partial ack received cwnd=cwnd-(currACK-prevACK)*mss+mss

DCTCP

A single flow needs C*RTT buffers for 100% TP
For large N flows C*RTT/sqrt(N) is enough
-Idea: React to ECN marks, every ECN mark cuts down window by 5% (TCP cuts by half regardless of number of marks)
-At switch mark packets when queue length > K
-At sender keep F=#markACK/totalACK, a=(1-g)*a+gF
-cwnd=(1-a/2)cwnd
Benefit: keep queue length short and TP high
Tradeoff: Convergence time is greater for new flows

TCP Losses

Block loss: lose a whole window of packets
Double loss: lose a retransmitted packet, protocol can't tell
-Solution: timestamp
Tail loss: one of the last packets of the stream is lost, not enough DUP ACK to trigger retransmission
-Solution: send dummy data (e.g. reiterated FIN)
PLATO: Send heartbeats interleaved to avoid RTO, to infer loss by 3 DUP ACK, heartbeat is rarely dropped

Traffic Scheduling: D3

Make network aware of flow deadlines
Prioritize based on deadlines
When capacity is greater than desired rates: deadline flows get desired rate + fair share, non-deadline get only fair-share
When capacity is not enough: greedily satisfy as many flows as possible according to request rates in order of arrival
-Need to modify hosts and switches, not backward compatible, no incremental deployment
-Not friendly with legacy transport protocols, running in parallel degrades performance

Traffic Scheduling - pFabric

-Prioritize packets based on remaining flow size
-pFabric switch: implement scheduling based on priority (send high priority first, drop low priority first)
-pFabric host: send/retransmit aggressively, use simple flow control (minTCP)
-Very small buffers, 2xLinkSpeedxRTT
-Worst case: small packets (64B), 51.2ns (64*8/10Gbps) to find min/max of 600 numbers with binary tree, 10 clock cycles, 1ns with current ASICs

Traffic Scheduling - pFabric 2

minTCP:
-Start at line rate, no RTO estimation, reduce window on packet drop, increase same as TCP (ss, CA)
Conclusions:
-Simple, yet near-optimal
-Requires new switches and minor host changes (clean-slate)
-Does not meet deadline requirements

Traffic Scheduling - Baraat

-Flow scheduling-> inefficient
-Priority scheduling -> does not meet deadlines
Idea: Task-aware scheduling
-Schedule tasks in Smart Priority Classes
-Switch maps flows to classes and handles heavy tasks
-Flows mapped to higher prio class get preference
-Flows with same priority class fair share
-TaskID is used as priority (FIFO)
-Heavy tasks are identified on the fly by byte count, upon exceeding threshold, task and immediately subsequent task are assigned same priority

Baraat features

Keep 3 counters: Total demand, total bytes reserved so far, number of flows in task
Also single aggregate counter for each link to keep track of BW allocations
Features:
-Schedule tasks, not flows
-FIFO-LM algorithm
-No need to know flow size
-New transport protocol
-Modifies switches and hosts
-Does not meet deadlines
-Reduces task completion time for partition-agg workflows compared to Fair share

Green DC

Minimize energy consumed by servers and cooling
-70-80% of total
-Consolidate workload to minimal set of servers and turn off unnecessary
-Consolidate workload based on locations to maximize efficiency of cooling
Minimize energy consumed by DC network (switches)
-10-20% of total
-Consolidate traffic to minimal set of paths and turn off switches/links

Green DC 2

Intra DC: dispatch loads to minimal servers and to cooler areas
Inter DC: dispatch loads to DC's with less energy cost or with renewable energy
JEC (Joint inter and intra)
-Considers variation of electricity prices and workload distribution on the efficiency of cooling systems
Random LB < Electricity InterDC < Cooling aware IntraDC < EIR+CIA < JEC

Elastic Tree

Power Knobs: vary link speed, disable links, disable switches, move workload
Goal:
-Turn off unneeded link and switch
-Create energy proportional DC network
Optimizer:
-Takes topology, routing restrictions, power models, traffic matrix
-Produces network subset and flow routes
Models:
-Formal: best quality, any topo, not scalable, input: Traff Matrix
-Greedy: good quality, any topo, scalable, traffic matrix
-Topo-aware: ok quality, structured topo, best scalability, port counters

CARPO

Considers BW demand variation over time
Elastic Tree might overestimate demand wasting power (average or peak, real demand is less)
-Use flow correlation (90 percentile data) to consolidate flows with low correlation using non-peak rate (low prob of peaking together)
-Minimize total power within a consolidation period based on traff correlation and non-peak data rate
-link rate adaptation for remaining links
Result: lowest power consumption and most savings, minor delay and drop degradation