Advanced Computer Networks

Author: Guanzhou (Jose) Hu 胡冠洲 @ UW-Madison CS740

Teacher: Prof. Ming Liu

This note includes the summary paragraph I wrote for the papers we reviewed this semester.

Network Architecture

NOW

Link: https://research.cs.wisc.edu/wind/Publications/now95.pdf

VL2

Link: https://dl.acm.org/doi/pdf/10.1145/1592568.1592576

This paper presents a new design of data-center networking architecture named *VL2* (which stands for Virtual Layer-2) that emphasizes assignment agility, performance isolation, and scalability. VL2 adopts a Clos interconnect topology, where aggregation switches and intermediate switches form a bipartite link-state network using *Locator Addresses* (LAs), while servers use *flat Application Addresses* (AAs) to locate any other server under ToR switches, decoupling addresses from locations to enable assignment flexibility. VL2 uses *Viliant Load Balancing* (VLB) and ECMP in flow granularity to randomly pick an intermediate switch for each connection between server pair. VL2 maintains a separate *Directory System* using Paxos to serve translations from AA to ToR LA. Evaluation results show that a VL2 prototype achieves comparable latency with traditional spanning-tree topology, while also providing uniform high capacity, fair load balancing, performance isolation, and fast convergence after failures.

BCube

Link: https://dl.acm.org/doi/pdf/10.1145/1592568.1592577

This paper presents BCube, a hypercube-based network architecture for modular datacenters (MDCs). The paper argues that existing datacenter network architecture designs, such as fat-tree and DCell, do not work well for MDCs, which might contain thousands of inexpensive commodity servers & switches each having a small number of ports and requires a low diameter to be packaged into shipping containers. A BCube$_k$ network of dimension k is recursively constructed out of BCube$_{k-1}$s, where servers use *gray-coding* to form a *hypercube*. The paper implements a BCube source routing (BSR) algorithm, which fully utilizes its high aggregate bandwidth of multiple disjoint paths and balances load using adaptive probing. Evaluation results show that a BCube prototype achieves high throughput for bandwidth-intensive applications and graceful degradation under server/switch failures.

Jupiter

Link: https://dl.acm.org/doi/pdf/10.1145/2829988.2787508

This paper is a retrospective on the design and deployment of five generations of datacenter network at Google, from Firehose 1.0 (2005) to Jupiter (2012). The paper elaborated on three main ideas. First, cost-effective scalable networks can be built using three-stage Close topologies out of commodity switches. Second, a top-down management approach viewing the datacenter as a static topology and distributing a centralized configuration to all switches proves to be simpler and more efficient compared to decentralized peer discovery protocols, sharing similarities with SDNs. Third, modular hardware and software design allows efficient and robust deployment at campus scale.

Routing

BGP Instability

Link: http://conferences.sigcomm.org/sigcomm/1997/papers/p109.pdf

This paper presents a statistical analysis of Internet BGP routing traffic measured at major US BGP exchanges around 1996. Results reveal several interesting characteristics of BGP routing *instability*. First, the majority 99% of routing traffic is *pathological* (i.e., redundant) and does not reflect topological changes. Second, the rest of routing messages, though having smaller volume, reflect *forwarding instability* and *policy fluctuation* and have a significant impact on BGP instability. Third, detailed analysis shows that instability is well distributed across AS's, and both instability and redundant routing information exhibit temporal bursts and periodical behavior.

Chord

Link: https://pdos.csail.mit.edu/papers/chord:sigcomm01/chord_sigcomm.pdf

The paper presents Chord, a decentralized lookup service based on consistent hashing that is highly scalable and supports dynamic membership changes. On a consistent hashing address space of size 2^m, Chord maintains a *finger table* of size m per node n, pointing to the successor of address $n + 2^i, 0 \leq i < m$, instead of an impractical full list of all N nodes. As a tradeoff, a Chord lookup could contain multiple hops, bounded by $O(\log N)$ in the worst case. Chord supports concurrent membership changes where nodes are joining/leaving the network and guarantees that the finger table state on all nodes will eventually converge (except in the case of partitioning) through a periodic stabilization algorithm. Chord tolerates node failures by having each node store a list of r successors instead of just one for redundancy. Simulation and evaluation results show that Chord can scale to a large number of nodes, support efficient lookups, and have graceful degradation of availability in the case of node failures and membership changes.

Flow Scheduling

Hedera

Link: https://www.usenix.org/conference/nsdi10-0/hedera-dynamic-flow-scheduling-data-center-networks

This paper presents Hedera, a datacenter network control plane that realizes dynamic flow scheduling for multi-root tree topology. Datacenter networks should be able to exploit multi-pathing and adapt to dynamic traffic patterns, yet existing hash-based ECMP/VLB approaches do not respond to congestion. The paper introduces a control plane, which connects a central scheduler to all the switches, for monitoring and estimating the demand of all flows and twisting their paths for load balancing. Two example scheduling policies, `Global First Fit` and `Simulated Annealing`, are proposed and evaluated. Results show that Hedera brings better bisection bandwidth than vanilla ECMP with reasonable out-of-band control overhead.

pFabric

Link: https://web.stanford.edu/~skatti/pubs/sigcomm13-pfabric.pdf

The paper presents pFabric, a minimal priority-based datacenter network transport design that decouples flow scheduling and rate control. pFabric annotates each packet with a priority number that reflects the scheduling goal, such as remaining flow size for minimizing flow completion time (FCT) or distance to deadline for maximizing number of deadlines met. pFabric switch implements a very simple scheduling algorithm, which always sends out the highest-priority packet in queue and always drops the lowest-priority packet when queue is full. pFabric switch controls its sending rate using a simple TCP/reno-like algorithm, which does not rely on any information about the flows. Simulation results show that, despite its extreme simplicity and conciseness, pFabric can achieve near-optimal FCT.

PIFO

Link: https://dl.acm.org/doi/pdf/10.1145/2934872.2934899

This paper presents a new programmable packet scheduler design using the *push-in-first-out* (PIFO) abstraction. PIFO switches allow a *packet transaction* to be executed upon the enqueuing of each new packet to decide its rank and to put it at the correct position in queue, while always dequeuing the packet at the head of queue. The simplicity of PIFO primitives makes it possible to synthesize hardware implementations that operate at line-rate performance. At the same time, by having a tree of PIFO queues and allowing optional *shaping transactions*, the PIFO abstraction is powerful enough to compose most scheduling algorithms, covering hierarchical algorithms and non-work-conserving algorithms.

Load Balancing

CONGA

Link: https://dl.acm.org/doi/pdf/10.1145/2619239.2626316

This paper presents CONGA, a distributed, in-network, congestion-aware load balancing technique for datacenter networks. CONGA operates in an overlay network, e.g., VXLAN, where all pairs of leaf ToR switches have (maybe virtual) "tunnels" between them so that each leaf knows the destination leaf of each packet and can carry an additional overlay header with the packet. As the packet traverses through the spine network, the overlay header is marked to carry the load on the most congested link on its path, measured in discounting rate estimator (DRE). Receiver maintains a Congestion-From-Leaf table and seeks opportunity to piggyback the load information to the next packet back to sender. Sender updates its Congestion-To-Leaf table with the load information and chooses the best uplink for a new flowlet. Evaluation results show that CONGA maintains low FCT even with significant load invariance.

Duet

Link: https://engineering.purdue.edu/~ychu/publications/sigcomm14_duet.pdf

This paper presents Duet, a data center load balancer design that combines hardware and software muxes to provide low latency, high scalability, as well as high availability. Duet first leverages idle resources on switches, especially empty entries in forwarding tables, ECMP tables, and tunneling tables to allow switches to function as low-latency, low-cost hardware muxes (HMux). VIPs are partitioned across multiple switches, where each switch stores the VIP-DIP mappings for all VIPs assigned to it. For better scalability and fault tolerance, Duet then adopts a software mux (SMux) cluster as a backup, storing all VIP-DIP mappings. A greedy VIP assignment algorithm runs periodically and tries to maximize resource utilization.

Resource Sharing

FairCloud

Link: https://www.mosharaf.com/wp-content/uploads/faircloud-sigcomm12.pdf

The paper presents a formal analysis on the sharing behavior of network links in a cloud setting, where multiple tenants each have a collection of VMs placed across nodes communicating with each other. The authors propose three fundamental requirements for network sharing: *min-guarantee*, *high utilization*, and *network proportionality*, and show that network proportionality is mostly exclusive from the other two requirements, so that it is impossible to have a perfect sharing policy, exposing a tradeoff space. The paper analyzes the behavior of traditional policies as well as proposes three new policies: PS-L, PS-P, and PS-N, that further explore the tradeoff space. Simulation results are presented to back up their formal analysis.

DRF

Link: https://www.usenix.org/conference/nsdi11/dominant-resource-fairness-fair-allocation-multiple-resource-types

The paper explores a problem that was not sufficiently addressed by previous work: fair allocation of multiple resources. The authors propose a new metric, *Dominant Resource Fairness* (DRF), that defines max-min fairness across users based on their dominant resource type. Analysis shows that DRF is *sharing incentive*, *strategy-proof*, *envy-free*, and *Pareto efficient*. They implement DRF scheduling in Mesos and show that DRF improves both CPU & memory resource utilization & fairness under a real Facebook Hadoop workload.

Programmable Networks

Ethane (SDN)

Link: http://cs.brown.edu/courses/csci2950-u/s14/papers/Casado07Ethane.pdf

The paper presents Ethane, a holistic authentication network design that demonstrates the early idea of software-defined networking (SDN). In Ethane, network access policies are managed by a central controller that installs flow table entries in dumb switches, in contrast to previous approaches such as using distributed smart switches or dedicated middleboxes. The Ethane controller maintains a global view of all bindings of switch port to MAC, MAC to IP, IP to host, and host to user registered in the network. All Ethane switches are just simple switches with a flow table data-plane, each establishing a secure channel with the controller for control-plane messages. The controller could be replicated for better fault-tolerance and scalability. A real implementation and deployment of Ethane at Stanford shows that this centralized design is simple, manageable, and effective, while being able to scale to a reasonably large campus network.

Open vSwitch

Link: https://www.usenix.org/conference/nsdi15/technical-sessions/presentation/pfaff

This paper presents the design and implementation of *Open vSwitch*, a software virtual switch for hypervisors. Open vSwitch targets a cloud-based virtual networking scenario, where end hosts are numerous VMs and their first-hop switches are virtual switches provided by the hypervisor software. The paper focuses on three major differences between virtual switches and traditional switches, and describes how Open vSwitch addresses these differences to provide good performance, flexibility, and manageability:

1. Virtual switches cannot assume dedicated hardware and must share computation resources with host applications, therefore Open vSwitch chooses to optimize for common-case lookup operations instead of worst-case guarantees. This leads to the design of splitting the switch functionality into a userspace component and a kernel datapath module, and applying flow caching to significantly improve lookup performance for subsequent packets.
2. VMs are numerous, so the network undergoes constant changes of flow states. This leads to the design of using a hashtable-based tuple space search algorithm for the lookup operation, because it has $O(1)$ complexity for updates.
3. For better manageability and easier addition of new switch features, Open vSwitch adopts the SDN approach with the OpenFlow model.

RMT

Link: https://dl.acm.org/doi/pdf/10.1145/2486001.2486011

The paper presents a practical hardware design of Reconfigurable Match Tables (RMT) in programmable switches for SDN. RMT extends the current OpenFlow "Match-Action" model to allow arbitrary packet header field definitions, flexible match table pipeline topologies, and easy addition of new actions. It does so through the following techniques. First, RMT decouples logical matching stages from physical pipeline stages and allows run-time mapping between the two. Second, RMT configures memory blocks per stage dynamically across parsing, matching, and actions, and allocates memory at a fine granularity. Third, RMT defines a minimal yet complete ISA for composing VLIW actions. Analysis shows that RMT brings high programmability with less than 15% extra cost over current commodity switch hardware.

AccelNet

Link: https://www.usenix.org/conference/nsdi18/presentation/firestone

The paper presents Azure AccelNet, an FPGA-based SmartNIC offloading solution by Microsoft that offers both good programmability and high, scalable performance. The main goals of AccelNet are to 1) reduce the CPU burden of increasingly complex host SDN networking stack in VM networking, 2) take advantage of the latency and throughput provided by SR-IOV NIC hardware, 3) maintain host SDN programmability, and 4) retain serviceability, flexibility, and

scalability. AccelNet meets these goals by integrating an FPGA-based SmartNIC on the path between host CPU & NIC and TOR, which implements a hardware-programmable General Flow Table (GFT) engine that enforces custom data-plane filtering rules. Control-plane logic remains in the host software stack. Data-plane traffic uses SR-IOV bypassing to NIC, which relays packets to the FPGA. AccelNet has been deployed at large scale in Azure and has yielded great improvement in performance while providing the same level of programmability as host SDN and introducing reasonable development difficulty.

Congestion Control

XCP

Link: https://dl.acm.org/doi/pdf/10.1145/633025.633035

The paper presents eXplicit Control Protocol (XCP), an ECN-based transport-layer protocol that improves the performance, stability, and fairness of TCP in high-bandwidth and long-latency network. XCP extends ECN to encode three congestion information fields in the header. In particular, sender puts its current `cwnd` size and `rtt` estimation, and fills the feedback field with a desired increase of rate proportional to the spare network bandwidth. Routers overwrite the feedback field if it is experiencing congestion to tell the source to adjust its rate proportionally to the level of congestion measured from queue lengths. The feedback is further divided fairly across packets. Analysis and simulation experiments show that XCP improves bandwidth utilization for large BDP networks, improves stability against bursts, provides fair, and decreases packet drops, without introducing too much complexity.

DCTCP

Link: https://people.csail.mit.edu/alizadeh/papers/dctcp-sigcomm10.pdf

The paper presents Data Center TCP (DCTCP), a new ECN-based congestion control protocol that adjusts congestion window size proportionally to the extent of congestion. DCTCP is simple and builds upon standard ECN/AQM mechanisms, such as RED, where routers mark the CE bit of a packet if its queue is above certain threshold and the receiver conveys the CE bit back to the sender through ECN-ACKs. The sender maintains a sliding-window estimation of level of congestion, $\alpha = (1-g)\alpha + gF$, where F is the fraction of CE-marked packets over a window and g is a constant parameter. It then adjusts its congestion window size proportionally to α, instead of simple AIMD. Analysis results show that DCTCP maintains low queue occupancy, guarantees low latency for interactive flows without sacrificing throughput, and improves stability.

Network Monitoring

OpenSketch

Link: https://www.usenix.org/conference/nsdi13/technical-sessions/presentation/yu

The paper presents OpenSketch, a software-defined network measurement architecture that supports a wide variety of measurement tasks and runs efficiently on commodity switch hardware. Previous flow-based measurement solutions such as NetFlow consume too many resources, while sketch-based measurement solutions are too ad-hoc. OpenSketch adopts the SDN idea from OpenFlow and splits measurement tasks into control plane and data plane. The switch implements a 3-stage programmable pipeline: hashing, classification (matching), and counting, which the authors prove to be general enough to support most measurement sketches and is also supported by current commodity hardware. The software controller provides a measurement library for operators to write simple measurement programs and auto-generate desired sketches and resource allocation policies. A prototype based on NetFPGA shows that OpenSketch consumes little memory space and produces <2% error rate across five measurement tasks.

Minions (TPP)

Link: https://dl.acm.org/doi/pdf/10.1145/2619239.2626292

The paper presents Tiny Packet Programs (TPP), a new architectural design for fine-grained network monitoring by injecting tiny custom programs to be executed per-packet per-hop. On the hardware side, a TCPU is added to each Match-Action stage of the data plane pipeline, which is able to execute TPP programs written in a minimal set of 6 instructions, carried by each packet. The instructions have access to both the packet header and the switch memory. On the software side, end host prepends packets with a TPP header that contains the TPP program and the space for storing queried information, collects those information for packets received, and analyzes them for desired metrics.

End-Host Network Stack

PicNIC

Link: https://dl.acm.org/doi/pdf/10.1145/3341302.3342093

The paper presents PicNIC, an end-host virtual NIC solution that improves performance predictability and isolation across VMs. The paper focuses on the fundamental tradeoff between bandwidth utilization (efficiency) and performance predictability (isolation) in a virtualized network. Prior work mostly focused on rate-limiting techniques in the fabric and did not investigate end-host issues -- a bursty flow from a misbehaving VM could negatively impact the goodput, delay, and loss rate experienced by other VMs due to contention on end-host resources.

PicNIC proposes a software virtual NIC design that addresses end-host contention issues. For each VM, PicNIC guarantees a min- and max-bandwidth envelope SLO, a latency distribution SLO, and a minimal loss rate (if the VM is within its bandwidth envelope). The design of PicNIC can be split into three constructs: 1) *CPU-fair weighted fair queues* (CWFQs) on the receiver side that divide the ingress engine's compute capacity in proportion to VMs bandwidth SLOs, 2) *receiver-driven congestion control* that computes the rate limit of each sender, and 3) *sender-side admission control* that shapes traffic and puts backpressure to guest VMs' transport layer.

Snap

Link: https://research.google/pubs/pub48630/

The paper presents Snap, a microkernel-oriented userspace network stack developed by Google. Snap runs as a standalone userspace process implementing end-host network stack modules, which were previously supplied by a monolithic kernel. Applications do network I/O through IPC with the Snap process, instead of doing syscalls.

Internally, the Snap process maintains groups of engines, where each engine is a single thread function providing certain network functionality required by the applications. Snap intensively uses resource accounting mechanisms e.g. MicroQuanta to ensure fair execution across engines, as well as kernel-bypassing techniques to offload processing & data to NIC hardware. As a result, Snap brings both high development velocity, easier maintenance, as well as good performance, without losing centralized control.