

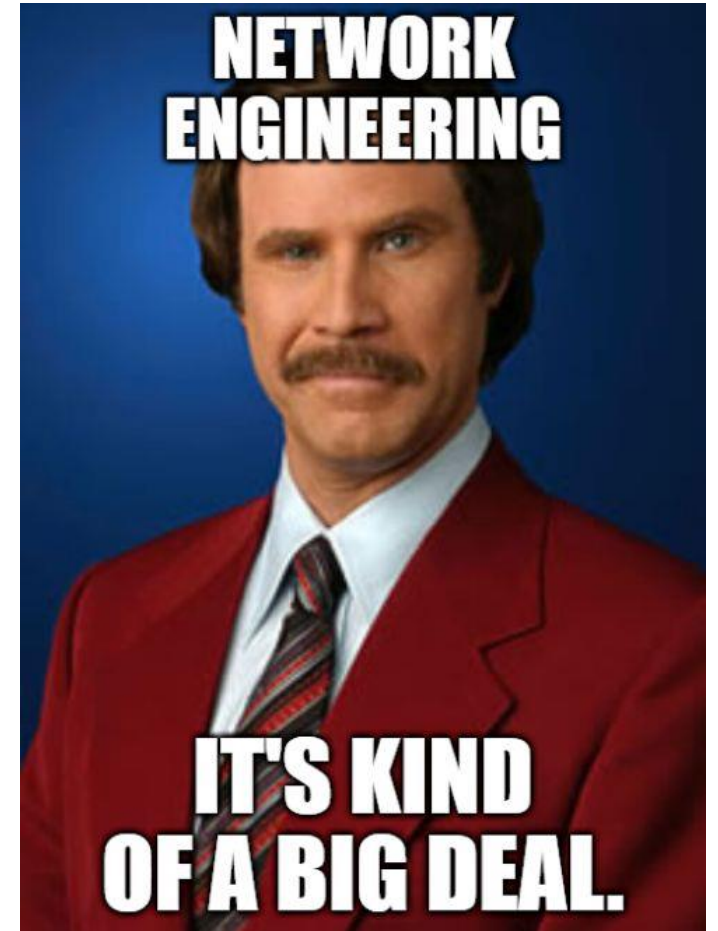


Getting 1+ Tbps from an x86 server

Branimir Rajtar, 5x9 Networks

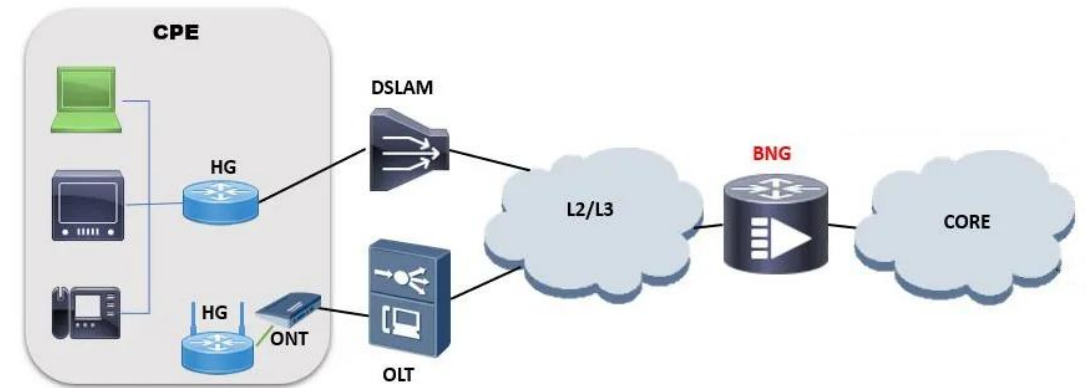
About the presenter

- Deploying traditional vendors 2010-2017
- Building a virtualized BNG since 2014
 - <https://github.com/OpenBRAS/OpenBRAS>
- Co-founder of 5x9 Networks, Chief Technology Officer
 - No coding, just getting smarter people to do the work
- Chair of Network Operator Group Croatia (NOG.HR)
 - It's mostly just about the beer after the meeting 😊



What is a BNG

- Broadband Network Gateway
- Terminates PPPoE/IPoE fixed-line user sessions
- Provides L3 connectivity to the Internet
- Enforces per user specific QoS profile and ACLs
- Responsible for AAA, LI, etc.

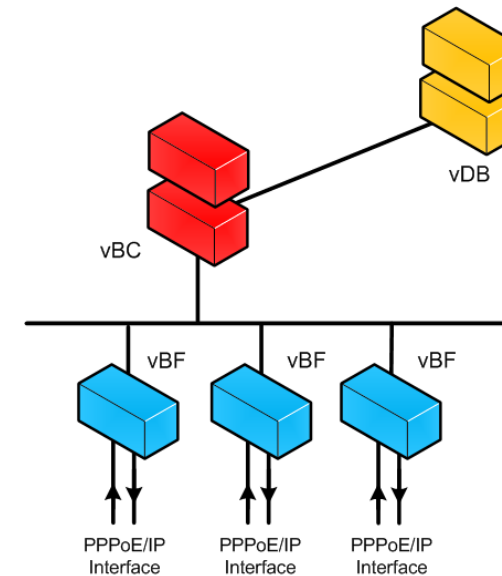


BNG market

	ASIC-based	White-label switch	x86
Throughput	Multi-terabit	Multi-terabit	~1 Tbit, getting better
Subscribers	*depends on chassis	~30k, getting better	~200k
Port count	highest	high	low, getting better
Power consumption	*depends on chassis	~250W	~600W
Development	proprietary	C (Broadcom API), P4	C (DPDK, eBPF, VPP)
Flexibility	Low	Broadcom-dependent, small memory footprint	High

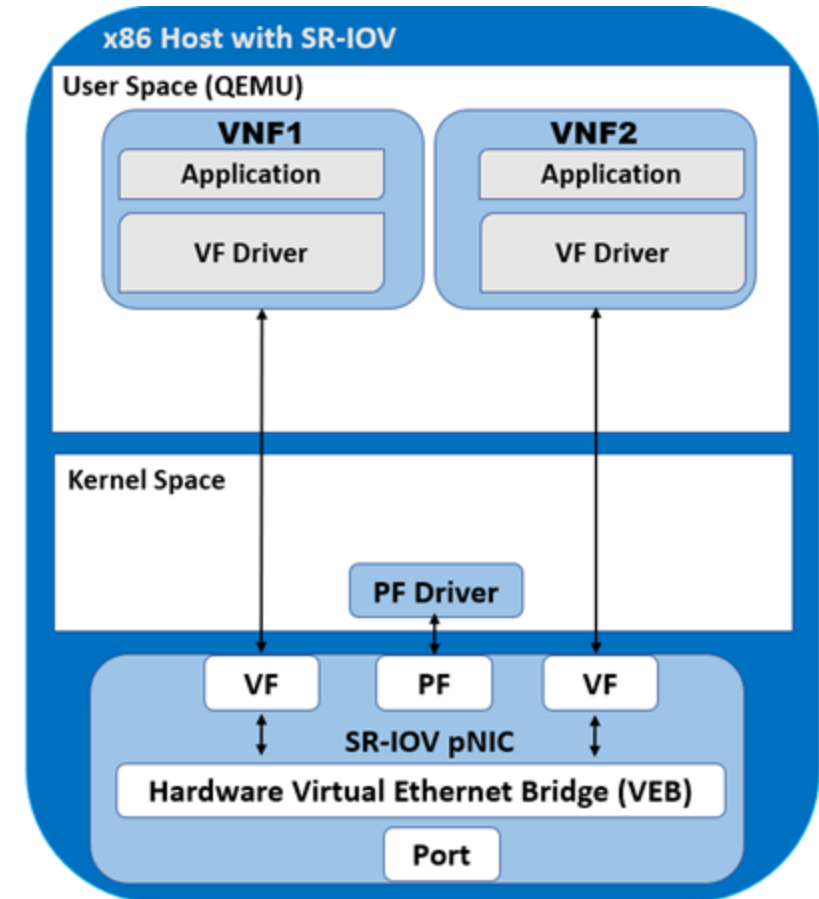
5x9 vBNG

- Virtualized CUPS system
 - Running in VM or container
 - Control and User Plane Separation
- Full automation built-in
- Three main components:
 - Dashboard – configuration, automation, GUI, API
 - Controller – Radius, IP allocation, routing, ...
 - Forwarder – PPPoE/IPoE termination, QoS, ACL, etc.
- Build for scalability and flexibility
- Forwarder relies on SR-IOV and Intel DPDK accelerations
 - More bandwidth → less servers → lower cost



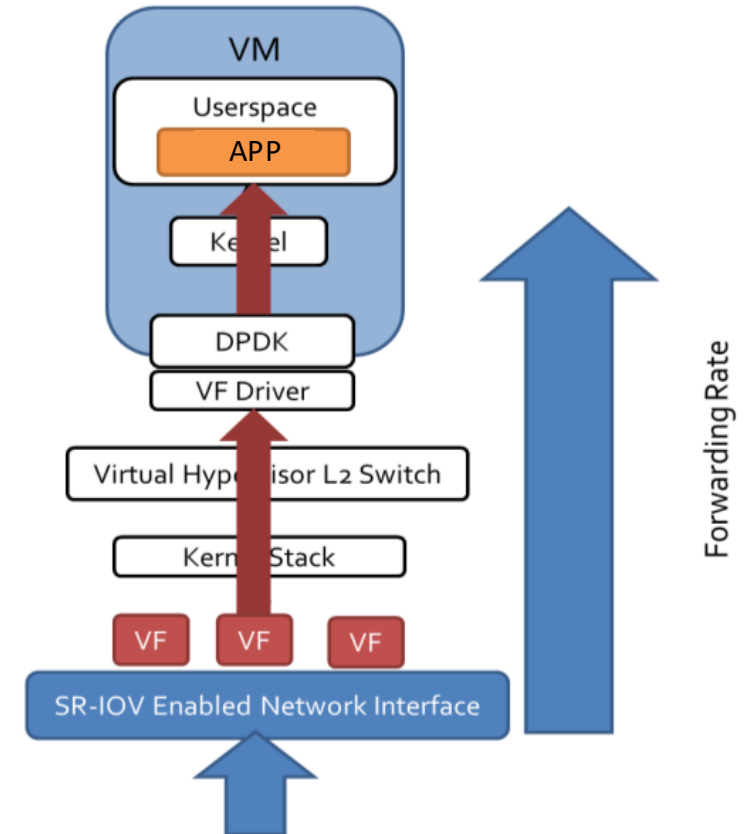
SR-IOV

- Single Root Input Output Virtualization
- Mechanism which presents virtualized physical device directly to user-space application
- Mainly used to go around host kernel and its drivers (slow!)
- Not supported by all hardware



Intel DPDK

- Data Plane Development Kit
 - Open source project
- A set of libraries, used to directly access network device skipping guest kernel and its drivers
- Works in “pull” mode
 - CPU constantly checks network device
 - Causes 100% load on CPU
- Network card must be supported by DPDK

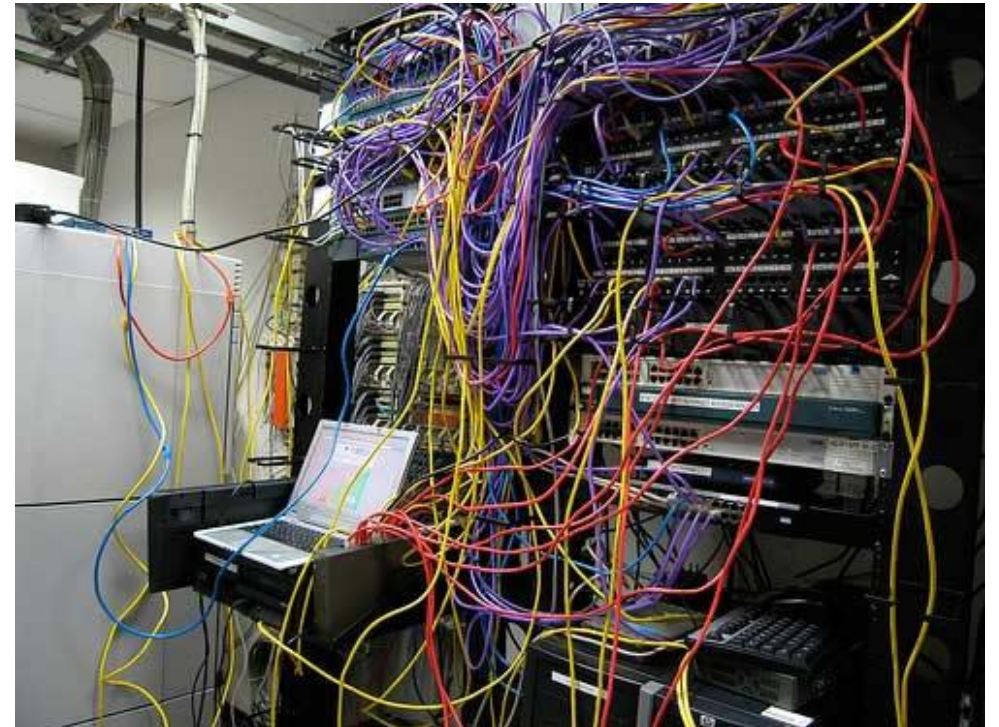


The “Small VM” Concept

- Initial concept was to make forwarder instance as small as possible
 - 4 vCPUs and 4 GB of memory per instance
 - Automated horizontal scaling approach used
- Pros
 - Optimal resource utilization
 - Each instance can have unique configuration
 - Easy to find virtual resources to start instance
- Cons
 - Large number of instances

Initial Performance

- 2x Intel Xeon Gold Gen2 CPU with 18 cores
- 16 Forwarders
- 40 Mpps no hQoS
 - 160Gbps for 500B packet size
- 26 Mpps using hQoS
 - roughly 35% performance penalty
 - 100Gbps for 500B packet size
- Intel reference document claims 100 Mpps per CPU
 - Newer hardware, simple in-out packet forwarding applications
 - Still significant performance difference

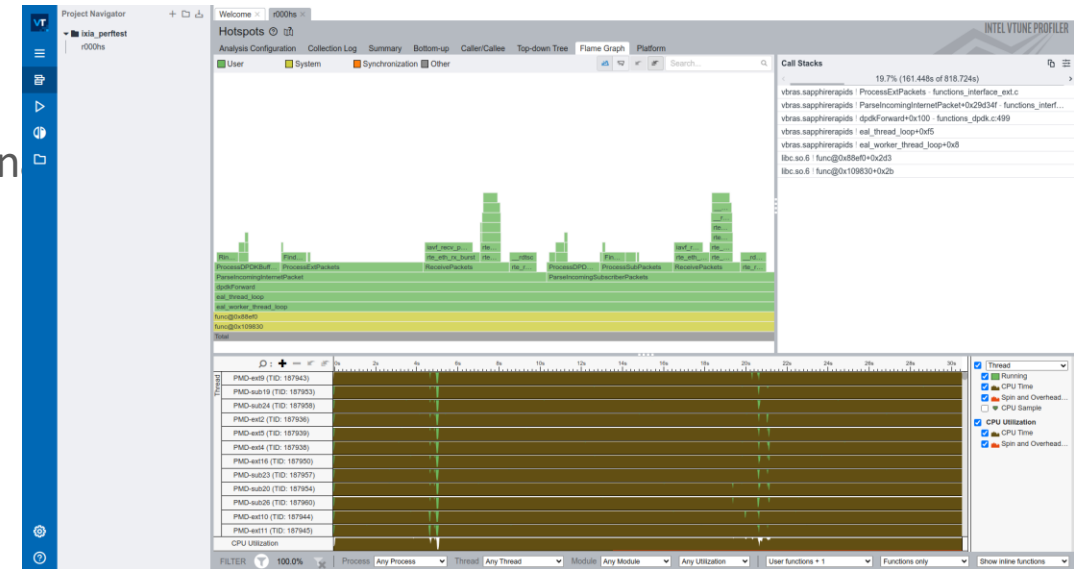


Quick Performance Wins

- Move to newer hardware
 - Optimize application for new HW
 - Initial 30% performance gain
- In-depth Intel DPDK tuning
 - Significant time spent to get deep understanding of underlying hardware
 - CPU options, PCIe communication, BIOS options, network cards tweaks
 - Additional 30% performance gain
- Required significant time to the change-testing cycle
 - Each change is tested to verify performance impact

Painful Deep Dive Into the Code

- Intel vTune application
 - Used for runtime in-depth code analysis using CPU intern
 - Provides detailed performance insights for the code
- Executed many code analysis using Intel vTune application
 - Identified “hot spots” in code (significant CPU wait)
 - Rewrite code to avoid waits (reorder instructions)
 - Rewrite loops to use multiple execution units
 - Reduce code – force compiler optimizations
 - Unfortunately, monitoring performance influences the performance itself
- Additional 20% performance gain



Moment of Revelation

- In the end, main cause was CPU cache misses and waits
- High count of memory object used a lot of CPU cache memory
 - CPU ended constantly pulling and pushing data from main memory
- Memory data changes caused cache stalls
 - Every change needs to be flushed back to main memory
 - Memory page (block) is 4k bytes!
- Cache alignment was not optimal
 - CPU reads a “cache line” (in most cases 64 bytes)
 - Single data object must be fitted in cache line

Post Revelation Actions

- Major code rewrite
 - Separate read and write data sections
 - Separate frequently and occasionally used data structures
- Identification of additional PCIe bus bottleneck
 - Addressed by grouping and carefully scheduling PCIe transactions (DMA)
- Introduction of more advanced hashing algorithms
 - Significantly reduced lookup cycles (to just 1)
 - Significantly reduced in-memory routing table footprint (90% reduction)
- Recognition of “Small VM” concept as inefficient
 - The same information is stored multiple times in the cache
 - Routing tables, interface tables, etc

The “Big VM” Concept

- Combine multiple small VMs to avoid in cache data multiplication
 - Single VM utilizes the entire NUMA resources and all available CPU
 - QoS mechanism and route lookup improvements
- Now it became clear why other VNF vendors are using similar design 😊

Current Result

- 1x Intel Xeon Gold 6780E with 144 cores
- 1 Forwarder, 64k subscribers
- 200 Mpps, ACLs, no hQoS (hQoS is enabled automatically per-user)
 - 800Gbps for 500B packet size
 - 32GB of memory usage
 - hQoS would require more memory
 - Only 2.5MB for 900k routing table!
- **1,6 Tbps** when scaled to 2x CPU
 - Performance starts to drop off after >100k subscribers (up to 260k subscribers supported)
 - Around 30% less performance if hQoS is enabled for all users (not required!)
 - Also 30-40% less performance if NAT is used for all users

Lessons Learned

- Always use latest hardware
 - Each CPU generation is better than the previous
 - Higher performance, lower power usage
 - CPUs with more cache, higher frequency, more cores, etc.
 - Intel and Mellanox 100/400GE network cards
 - Number of PCI lanes and available slots might be an issue
- Embrace the change, (re)use what is good
 - Switch to “Big VM” concept
 - Can be flexible and use the number of available cores
 - Fits the available hardware – customers are often not flexible, they reuse what they have
 - Use “Small VM” concept where it fits best
 - Perfect for Edge Cloud deployment and small number of users

Lessons Learned

- Use the available tools
 - Open source, closed source, AI
 - AI can point in the right direction or give you a new idea
 - Lots of tools are available, spend some time on investigating
- Continuous optimization is a must
 - What gets measured gets improved
 - Curiosity, learning, mindset and attitude are the key

Further Reading

- Intel DPDK
<https://www.dpdk.org/>
- Vector Packet Processing – Fast Data Project
<https://fd.io/>
- Open source testing tools – Intel vTune (profiling), BNG Blaster, Trex (packet generators)
- Open source routing protocol suite – FRRouting

Thank you!
Q&A