BROCADE

SOUTHBOUND SDN API'S

Curt Beckmann, Brocade

IETF84 SDNRG

Say you want to revolutionize networking…

Perhaps you want to do something like this:

Create a framework to support external orchestration of network device behavior across many problem spaces, enable independent innovation and promote competition and increase interoperability so as to reduce vendor lock-in.

Image URL: http://en.wikipedia.org/wiki/File:Johannes_Adam_Simon_Oertel_Pulling_Down_the_Statue_of_King_George_III,_N.Y.C._ca._1859.jpg

Oh, *And* it would be really nice if…

- …we can leverage lots of the existing device hardware
  - Seems like a good idea if you really want SDN to go places,
    - Otherwise you'll need major vendor investment in specialized hardware with an uncertain market based on an unstable protocol. Seems dicey.
  - Now, you could just let the soft targets take off and get established
    - But those soft targets will likely gain traction in areas that are not the compelling niches for specialized hardware.
  - Anyway, there's good evidence that a lot of existing hardware is capable of doing interesting things
    - If we could just control it!

Image URL: http://en.wikipedia.org/wiki/File:Archimedes_lever_(Small).jpg

Also, we'll start with a Southbound API

Um…. Why? Good question. Glad you asked.

- One could argue that working top-down would deliver a better architecture
  - On the other hand, there's no specific "top" to attempt "top-down".
  - Instead, there are several interesting high level application spaces
  - So top-down architecture maps to "boil the ocean"
- Also: it's been more movement than project
  - Needed to be able to demonstrate things
  - SB API was essential.
  - NB API not required… depends on architecture

Image URL: http://ecx.images-amazon.com/images/I/51AEVWEHC0L._SS500_.jpg

“Classic” SDN picture

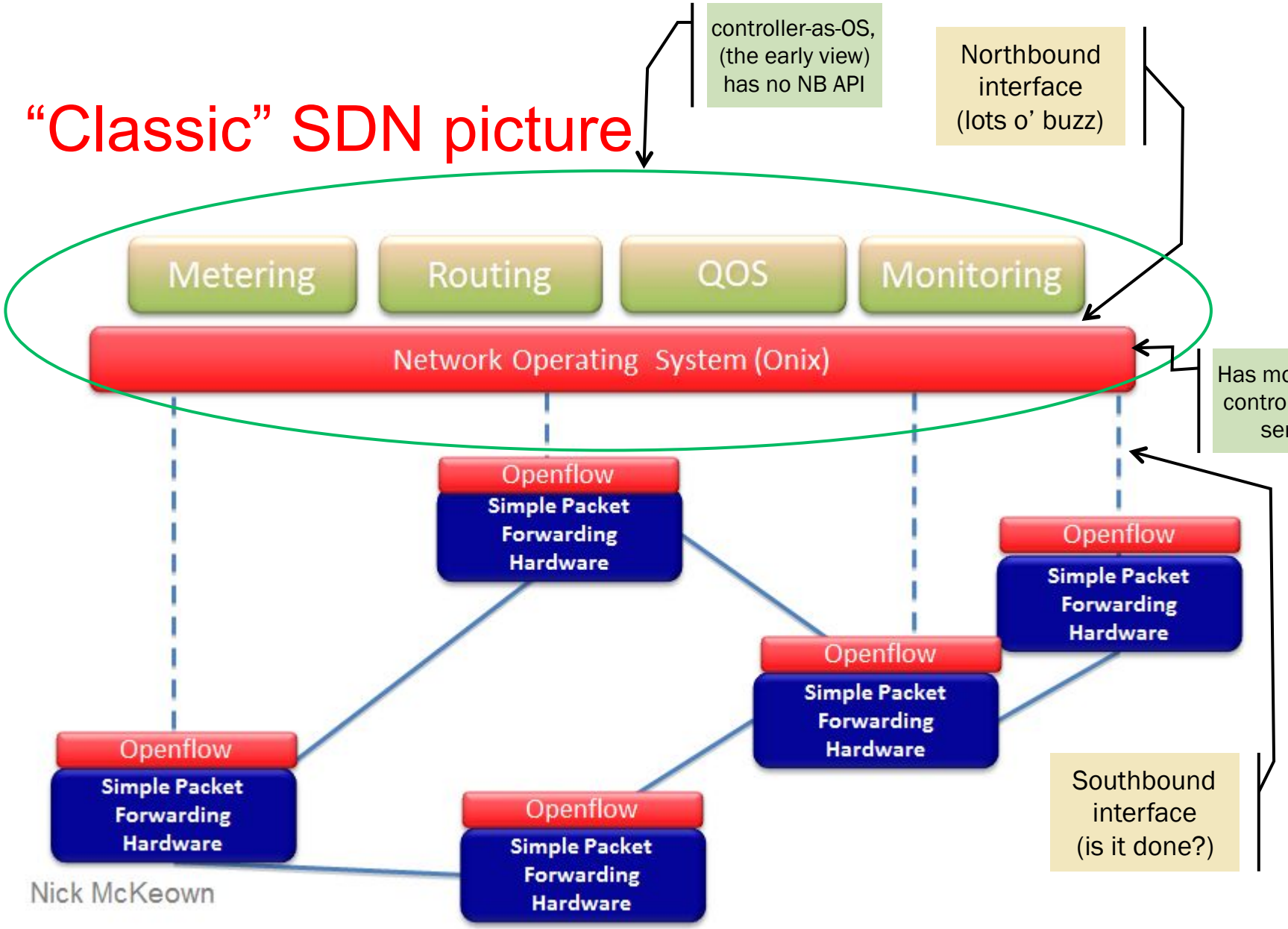


Image URL: http://perspectives.mvdirona.com/content/binary/NickMckeownSDN.jpg

So OpenFlow is a "Southbound API"

How do we define it?

- Side note: "API" is top-down (new thinking),
  - OF is technically a protocol spec (traditional bottom-up angle)
  - That's a just detail; they're both about abstractions
- What abstractions will we use in our protocol/API?
  - Should be driven by the problems. And we want to address a large number of problems (not just 1 or 2) for many reasons:
    - Get a big fan base, lots of members, economies of scale
    - If small #, opponents might solve small # of problems, steal thunder
    - Broad base improves chances of finding market-viable "killer app" sooner
  - But a large base does have other issues
    - For one, it makes it hard to reach consensus on the abstractions!
  - Okay, take a step back and ask: what are we trying to achieve?

Task: Orchestrate networking devices

One possible approach…

- We could manipulate the abstractions already used by network-based control protocols.
  - But there are issues:
    - No unified collection of abstractions, need a way to bridge from one framework to another
    - Today's most interesting target problem spaces are those where current protocols are failing to deliver what operators really need, so relying on existing abstractions would risk hitting the same roadblocks
    - And it would mean that existing vendors would have to hack (many?) existing control stacks
    - And it would mean that new vendors would be required to have (make or buy-and-hack) stacks

http://www.bach-cantatas.com/Pic-Bio-BIG/TS0-03[Sep2009].jpg

Orchestrate networking devices, Take 2

Another approach…

- Instead, aim at controlling device forwarding behavior more directly
  - This is the approach that OpenFlow opted for
- But: few established abstractions for low level forwarding behavior across a wide variety of protocols
  - Exception: TCAM seems to be pretty common

Proposal: define a common device model

Using TCAM as a foundation

- There are two obvious approaches.
  - A: Aim for a "common denominator" model, with features that all "modern" devices should have
    - Upside: most existing hardware would support this model
    - But!: such models would be limited. Many inexpressible behaviors. "common denominator" → "LEAST cd"
  - B: Create a "supermodel", that has every feature found on any device, plus a few more just in case
    - Upside: such models would be very powerful!
    - But!: No existing hw would support the full model.
      - And even supermodels won't do everything you want
    - Good news: Real apps only need subsets of supermodel to do cool stuff
    - New challenge: **Need to map** diverse subsets to real hw

http://blog.bitchingandjunkfood.com/wp-content/uploads/2012/03/Cindy-Crawford-bathing-suit-picture.jpg

OpenFlow has tried both

- OpenFlow 1.0 used Approach A, “least common D”
  - Result: “broadish” adoption, but real apps need extensions
- OpenFlow 1.1, 1.2, 1.3 use Approach B, “supermodel”
  - Result: Er, well, hard to tell since zero adoption so far
    - Many theories for why this is the case, but…
  - Maybe that’s not a coincidence?
    - Current OF framework burdens the Switch code with piecemeal mapping
    - OF1.3 provides a Table Features msg, but unclear how it helps mapping
- Is OpenFlow trying to support too many old targets?
  - No. Even new hw aligns well to some problems but not others
  - Narrowing hardware is not the fix

Is it just too hard?

Maybe the goals of OpenFlow too ambitious?

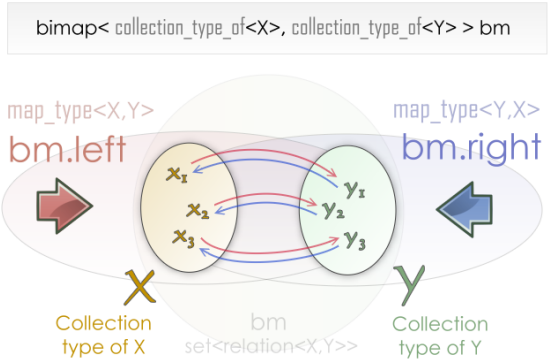
- Maybe.
- Or maybe there is some approach we've overlooked?
  - Well, of course there's another approach… See next slide.
- The thing is, any "single model" approach will either be:
  - Too basic and unable to do much, or
  - Super flexible, more capable than real-world devices → mapping
  - Or even both!
    - The full current OpenFlow model exceeds real world hardware*, and yet it cannot express many common functions present in existing hardware.

Image URL: http://professionaldestiny.com/wp-content/uploads/2009/11/pushing-rock-up-hill.jpg

*The current model can be implemented in many softer targets, which is highly relevant. The tradeoff is cost/ performance metrics differ by orders of magnitude. And the expressibility issues still exist.

Another approach

- The OpenFlow Future discussion group is in the process of proposing a new approach
  - Or maybe it's just a new variation on Approach B
- Instead of a single model, we modify the framework to allow for multiple models.
  - Q: Won't that take us down a "standards deadlock" path?
    - How does that help?! Isn't SDN supposed to accelerate innovation?
  - A: It helps if new models can be created without invoking the standards creation process
- New game: clearly specify ***how to describe*** a model
  - → market players get to pick and design the models they need



Mapping is key challenge

Requested behavior → target

- OpenFlow 1.x describes desired behavior via “match” and “action” fields in a “flowmod” message.
  - Flowmods create flow entries in a flow table
- OpenFlow 1.0 used a single flow table. Easy!
  - Flowmods describe end-to-end 1.0 behavior for matching flows.
  - Coders can deliver a priori mapping of all supportable flowmods.
- OF 1.1+ added 256 tables, “goto Table X” actions, etc.
  - Infinite (?) flowmod combos may yield desired platform behaviors
  - Flowmods are now partial behaviors instead of end-to-end
  - Coders must (somehow?) implement bit-by-bitmapping algorithm
  - Sadly, OpenFlow “primitives” map very poorly to platform internals

http://www.boost.org/doc/libs/1_41_0/libs/bimap/doc/html/images/bimap/extended.mapping.framework.png

Seems like rock and hard place?

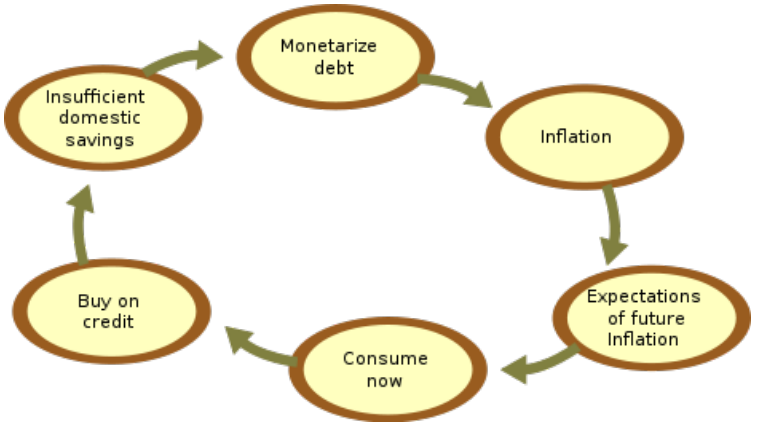
- We want flexibility
- Indeed, we want more flexibility than we have now!
- But flexibility makes for harder implementation
- Harder implementation impedes adoption
- And adoption is already problematic

- Seems like a vicious cycle. Are we stuck?

Image URL: http://freerisk.org/wiki/images/c/c9/468px-Vicious_circle_in_macroeconomics.svg.png

No, we're not stuck.

The vicious cycle is not hard-wired in



- The current framework has two arbitrary aspects that make things extra hard
  - In the current framework, the mapping logic only gets partial information in piecemeal fashion (those "flowmods")
  - Also, the mapping intelligence is required to reside on a switch and must solve the mapping very quickly at run-time
    - This is despite the fact that network operators would expect interoperability to be fully validated before run time. No surprises allowed.
    - If interoperability is resolved pre-run-time, then the mapping must already have been resolved.
    - If it has already been resolved once, why re-resolve it with each connection?

The way out

"What obstacles?"

- Provide information about the desired behavior at a higher level
  - Describe flow handling at the switch level instead of in incremental "flowmod" tidbits
- Share this information before run-time
  - Provide the information in the form of "well described abstractions"
  - Register unique IDs for the abstractions
  - Enable the controller and the switch to negotiate agreed ID's at run time
- In other words, break the vicious cycle by moving to a simpler framework

Photo URL: http://www.getsurrey.co.uk/news/s/2067511_car_crashes_through_wall_in_latest_village_accident

BROCADE

Questions?