A Softwarized Internet of Touch

Fernando Kuipers
Lab on Internet Science, Delft University of Technology
https://fernandokuipers.nl/

Joint work with:

- Niels van Adrichem
- Jorik Oostenbrink
- Mohammad Riftadi
- Belma Turkovic

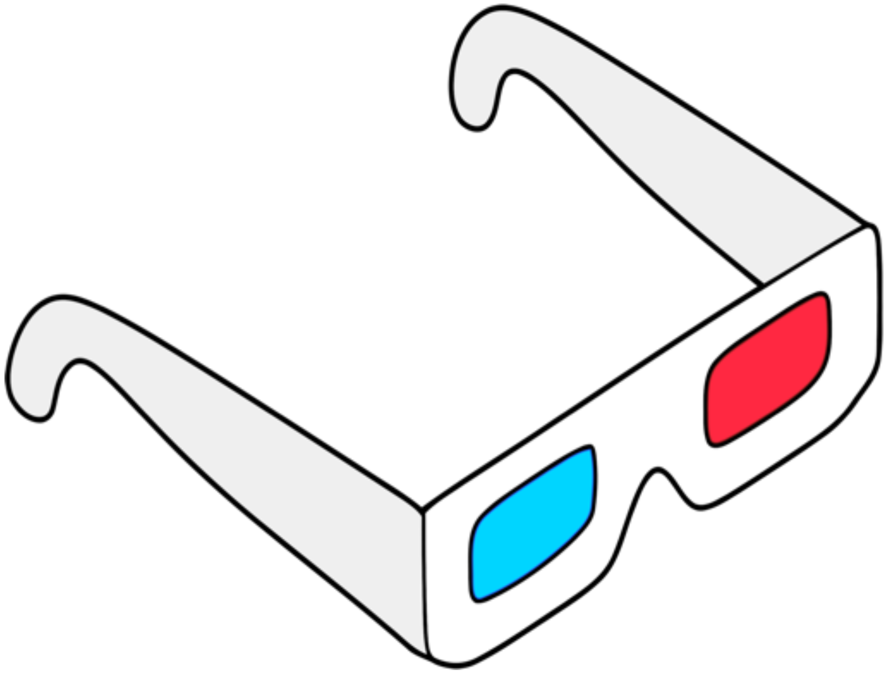
TU Delft

Do I∘T initiative

- Delft on Internet-of-Things
  https://www.tudelft.nl/iot/

- Mission:
  - Connect researchers & industry
  - Combine technology and creativity (within and beyond TU Delft)
  - Create a 5G+IoT field lab

30 years ago…



What if you could remotely control real or virtual objects in real time?

“The Tactile Internet”

TU Delft

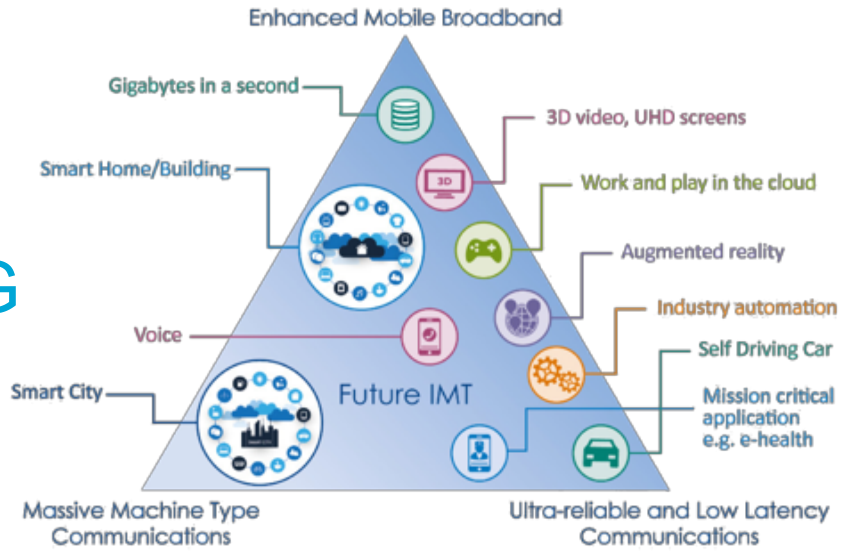

2
1
3


VR1280
Virtual Research

1 ms challenge

- Ultra-responsive (round-trip latency of 1 ms)
- Ultra-reliable (outage of about 1 ms/day)

5G



Source: https://www.etsi.org/images/articles/Future-IMT.png

Software-Defined Networking (SDN) to the rescue?

TUDelft

Traditional routing

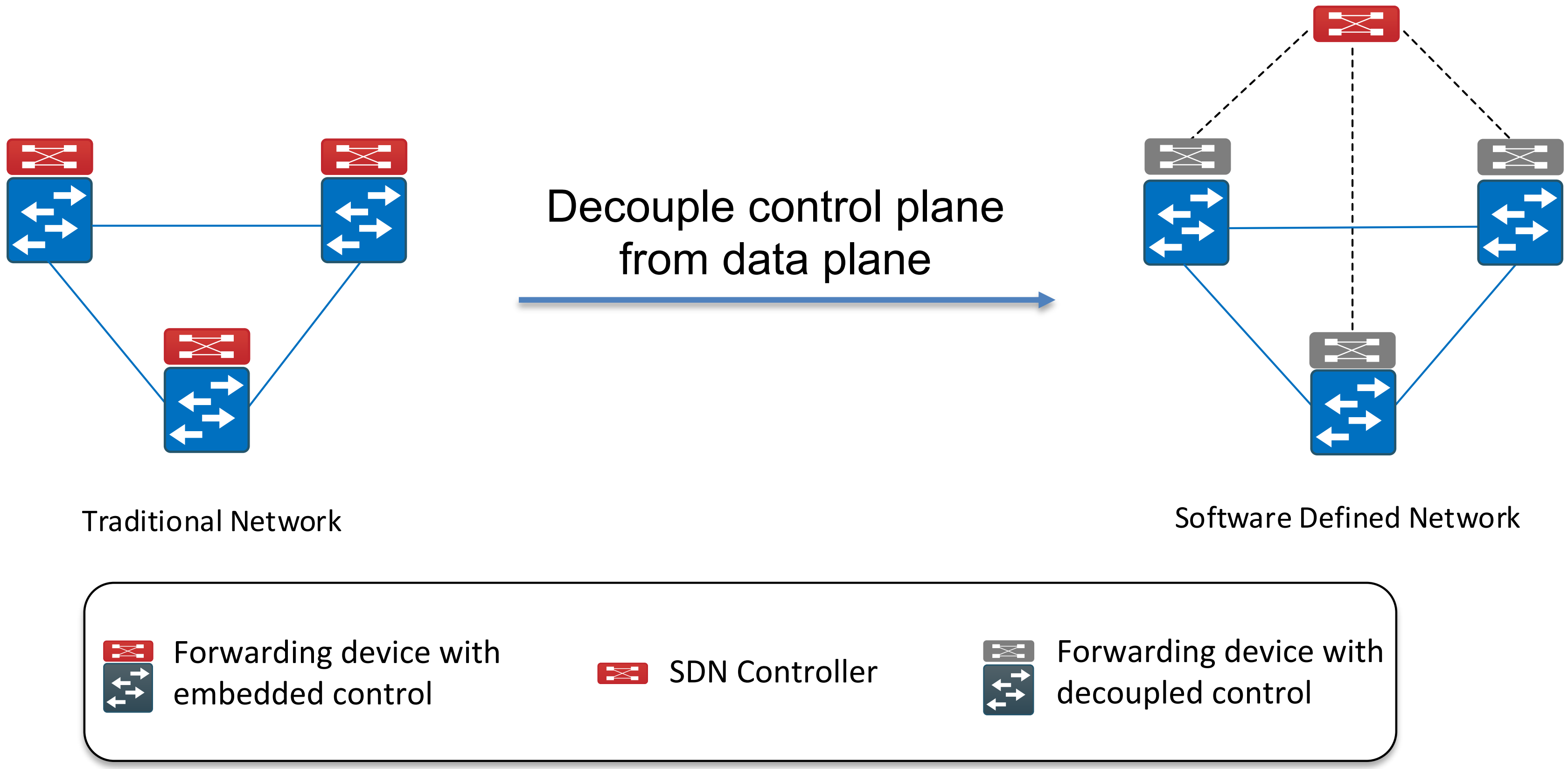
- Control Plane:
  - Maintains routing table
  - OSPF, BGP, …
- Data Plane:
  - Forwards packets

Source: https://commons.wikimedia.org/wiki/File:Cisco-rs1.jpg

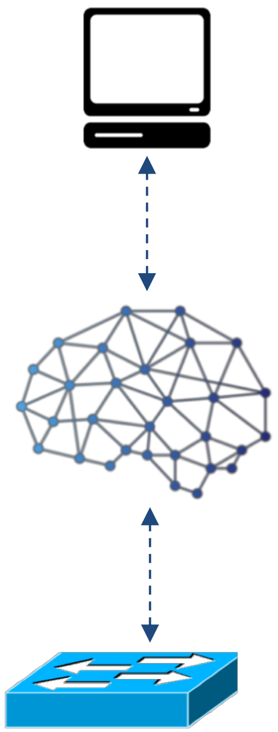
Disadvantages of traditional routing

- Difficult to add new functionality (proprietary software)
- Built on fixed-function hardware
- Complex network management
- Constant communication between routers

Software-Defined Networking



Advantages of SDN



- APPs: monitoring, security, TE, …
- Controller a.k.a. Network Operating System
  - Centralized decision making
  - Programmable

Benefit from open source solutions

- Switches
  - Only need to worry about forwarding
  - Reduced CapEx

Even open hardware: https://www.opencompute.org

OpenFlow

OpenFlow: Enabling Innovation in Campus Networks

Nick McKeown
Stanford University

Tom Anderson
University of Washington

Hari Balakrishnan
MIT

Guru Parulkar
Stanford University

Larry Peterson
Princeton University

Jennifer Rexford
Princeton University

Scott Shenker
University of California, Berkeley

Jonathan Turner
Washington University in St. Louis

This article is an editorial note submitted to CCR. It has NOT been peer reviewed.
Authors take full responsibility for this article's technical content.
Comments can be posted through CCR Online.

ABSTRACT

This whitepaper proposes OpenFlow: a way for researchers to run experimental protocols in the networks they use every day. OpenFlow is based on an Ethernet switch, with an internal flow-table, and a standardized interface to add and remove flow entries. Our goal is to encourage networking vendors to add OpenFlow to their switch products for deployment in college campus backbones and wiring closets. We believe that OpenFlow is a pragmatic compromise: on one hand, it allows researchers to run experiments on heterogeneous switches in a uniform way at line-rate and with high port-density; while on the other hand, vendors do not need to expose the internal workings of their switches. In addition to allowing researchers to evaluate their ideas in real-world traffic settings, OpenFlow could serve as a useful campus component in proposed large-scale testbeds like GENI. Two buildings at Stanford University will soon run OpenFlow networks, using commercial Ethernet switches and routers. We will work to encourage deployment at other schools; and We encourage you to consider deploying OpenFlow in your university network too.

Categories and Subject Descriptors

C.2 [**Internetworking**]: Routers

General Terms

Experimentation, Design

Keywords

Ethernet switch, virtualization, flow-based

1. THE NEED FOR PROGRAMMABLE NETWORKS

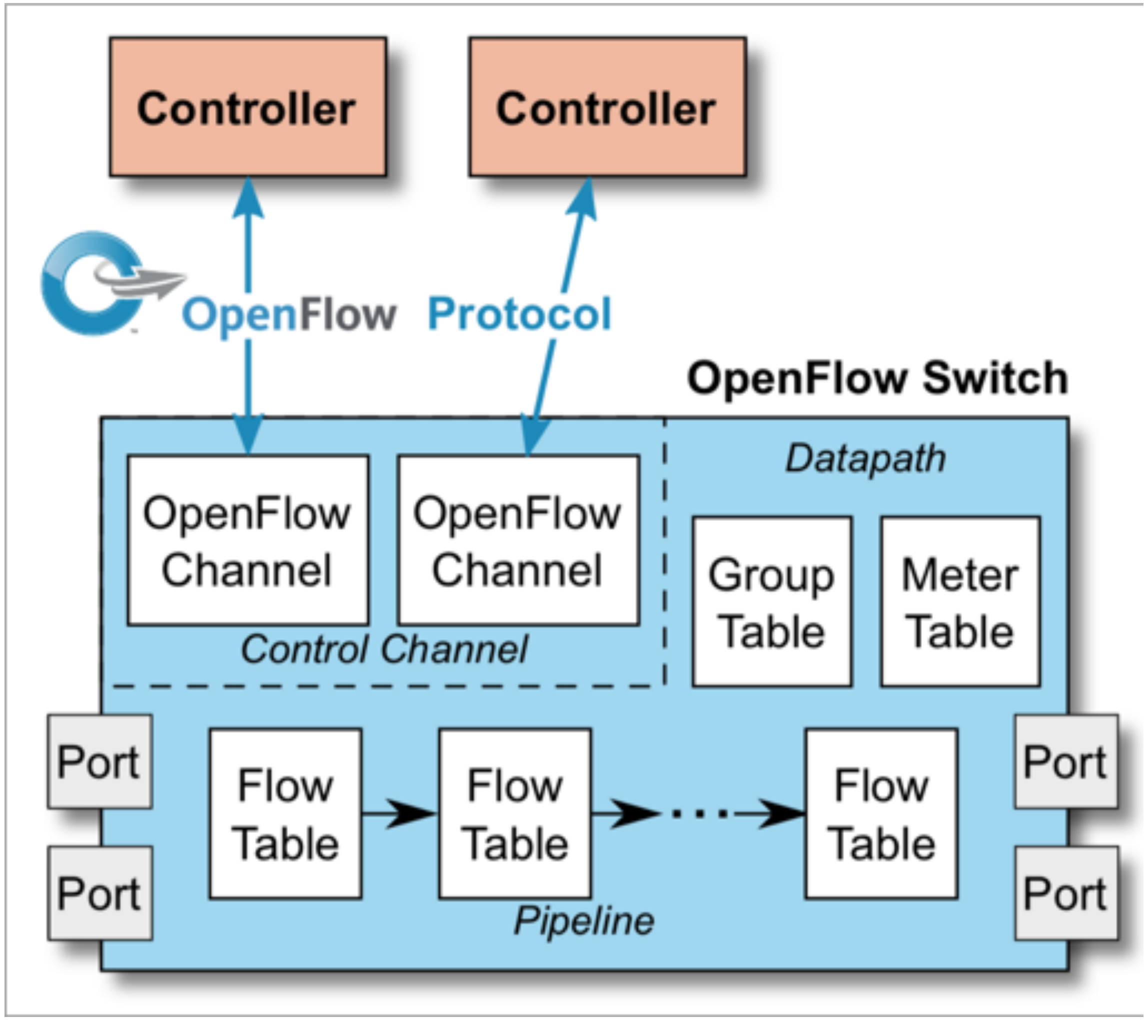
Networks have become part of the critical infrastructure of our businesses, homes and schools. This success has been both a blessing and a curse for networking researchers; their work is more relevant, but their chance of making an impact is more remote. The reduction in real-world impact of any given network innovation is because the enormous installed base of equipment and protocols, and the reluctance to experiment with production traffic, which have created an exceedingly high barrier to entry for new ideas. Today, there is almost no practical way to experiment with new network protocols (e.g., new routing protocols, or alternatives to IP) in sufficiently realistic settings (e.g., at scale carrying real traffic) to gain the confidence needed for their widespread deployment. The result is that most new ideas from the networking research community go untried and untested; hence the commonly held belief that the network infrastructure has "ossified".

Having recognized the problem, the networking community is hard at work developing programmable networks, such as GENI [1] a proposed nationwide research facility for experimenting with new network architectures and distributed systems. These programmable networks call for programmable switches and routers that (using *virtualization*) can process packets for multiple isolated experimental networks simultaneously. For example, in GENI it is envisaged that a researcher will be allocated a *slice* of resources across the whole network, consisting of a portion of network links, packet processing elements (e.g. routers) and end-hosts; researchers program their slices to behave as they wish. A slice could extend across the backbone, into access networks, into college campuses, industrial research labs, and include wiring closets, wireless networks, and sensor networks.

Virtualized programmable networks could lower the barrier to entry for new ideas, increasing the rate of innovation in the network infrastructure. But the plans for nationwide facilities are ambitious (and costly), and it will take years for them to be deployed.

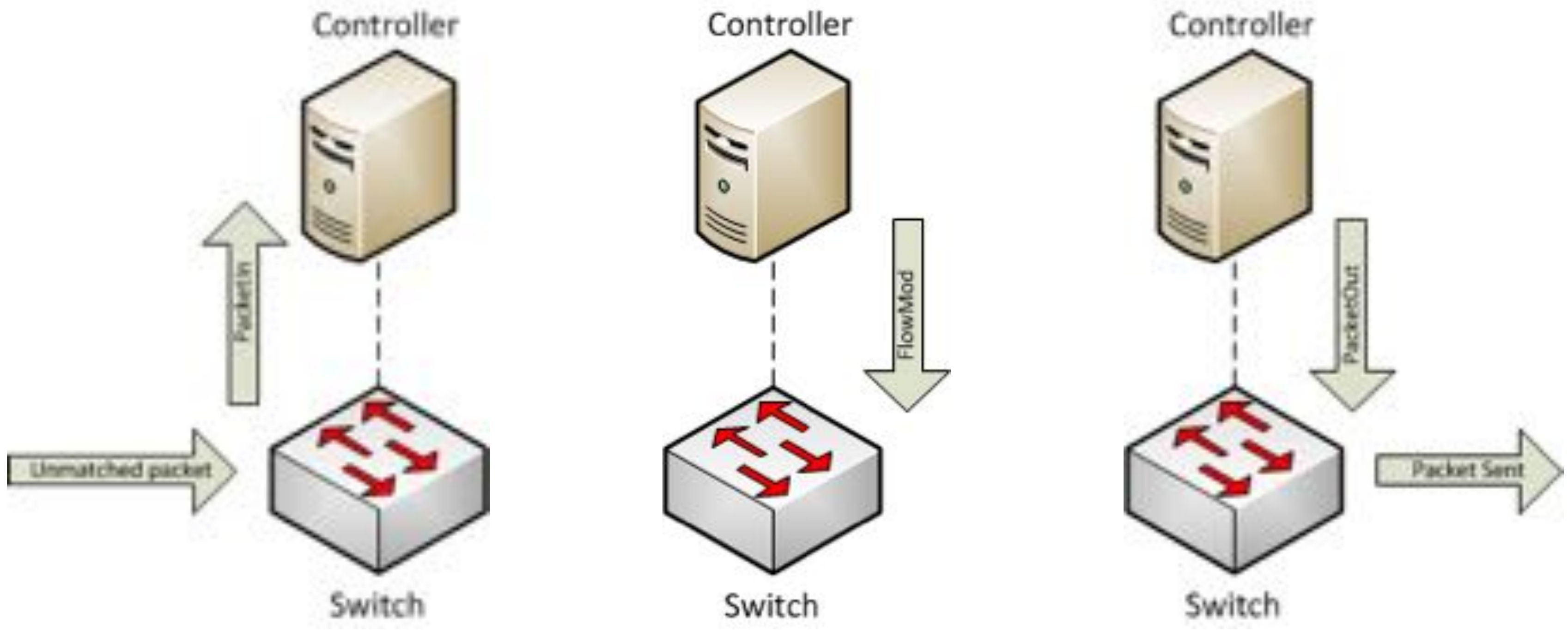
This whitepaper focuses on a shorter-term question closer to home: *As researchers, how can we run experiments in our campus networks?* If we can figure out how, we can start soon and extend the technique to other campuses to benefit the whole community.

To meet this challenge, several questions need answering, including: In the early days, how will college network administrators get comfortable putting experimental equipment (switches, routers, access points, etc.) into their network? How will researchers control a portion of their local network in a way that does not disrupt others who depend on it? And exactly what functionality is needed in network



Source: OpenFlow Switch Specification v1.5.1

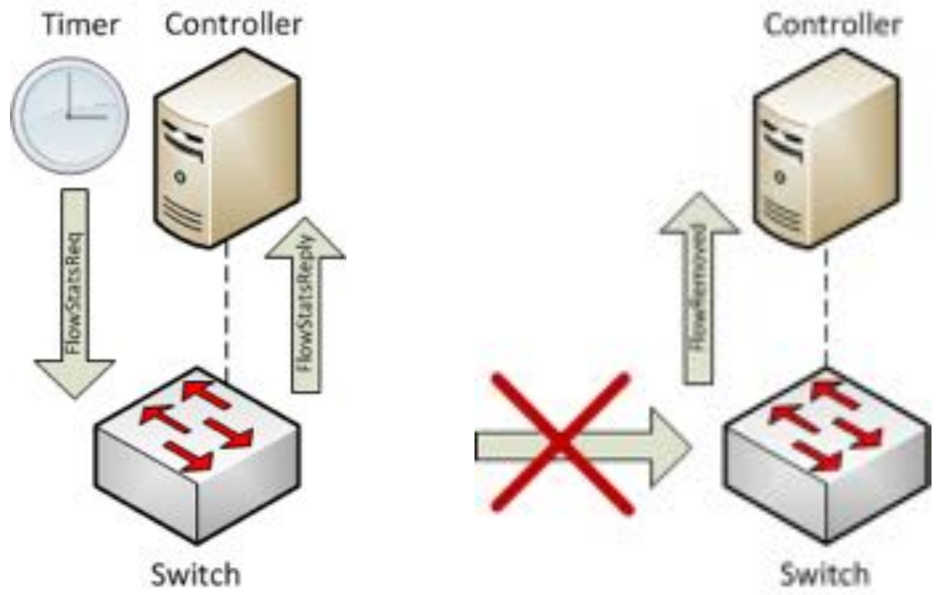
Installing flow rules



OpenNetMon: network telemetry

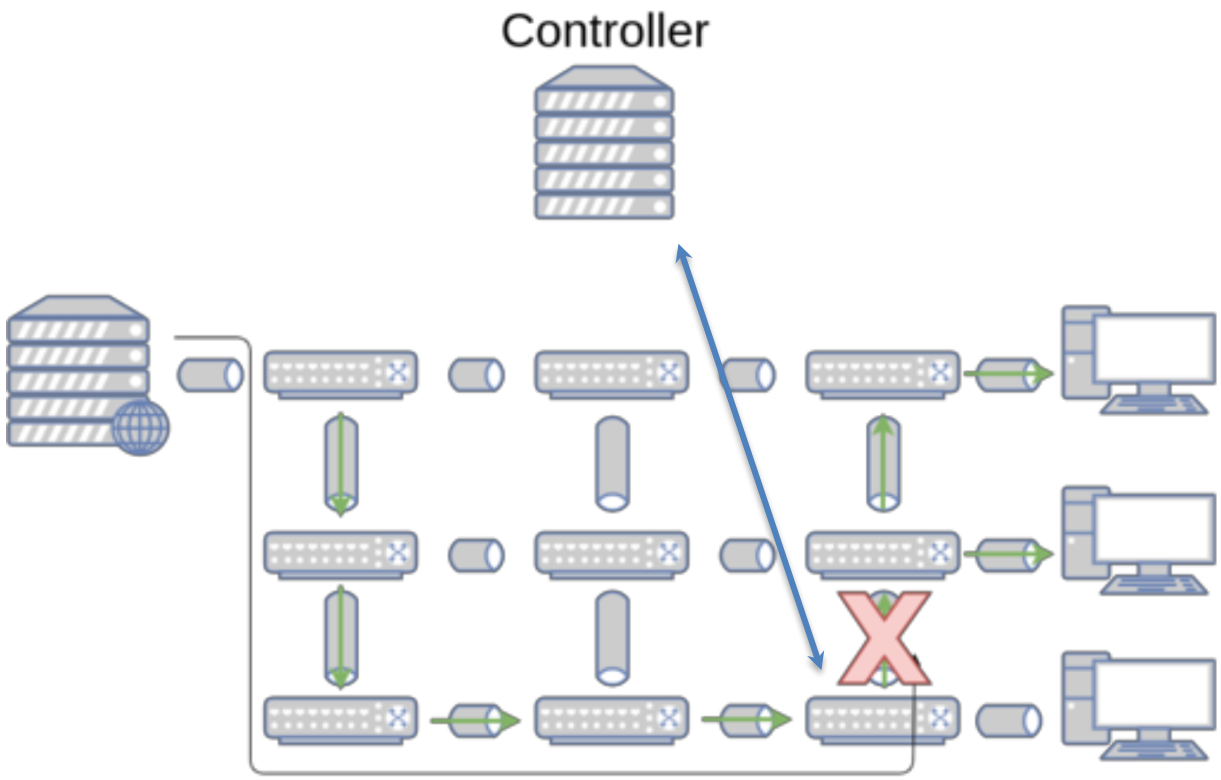
- Per-flow counters
  - Packet counters (PC)
  - Byte counter (BC)
  - Flow duration (FD)

- Throughput: $\frac{\Delta BC}{\Delta FD}$
- Packet loss: PC_{start} - PC_{end}
- Delay: timed probe packets



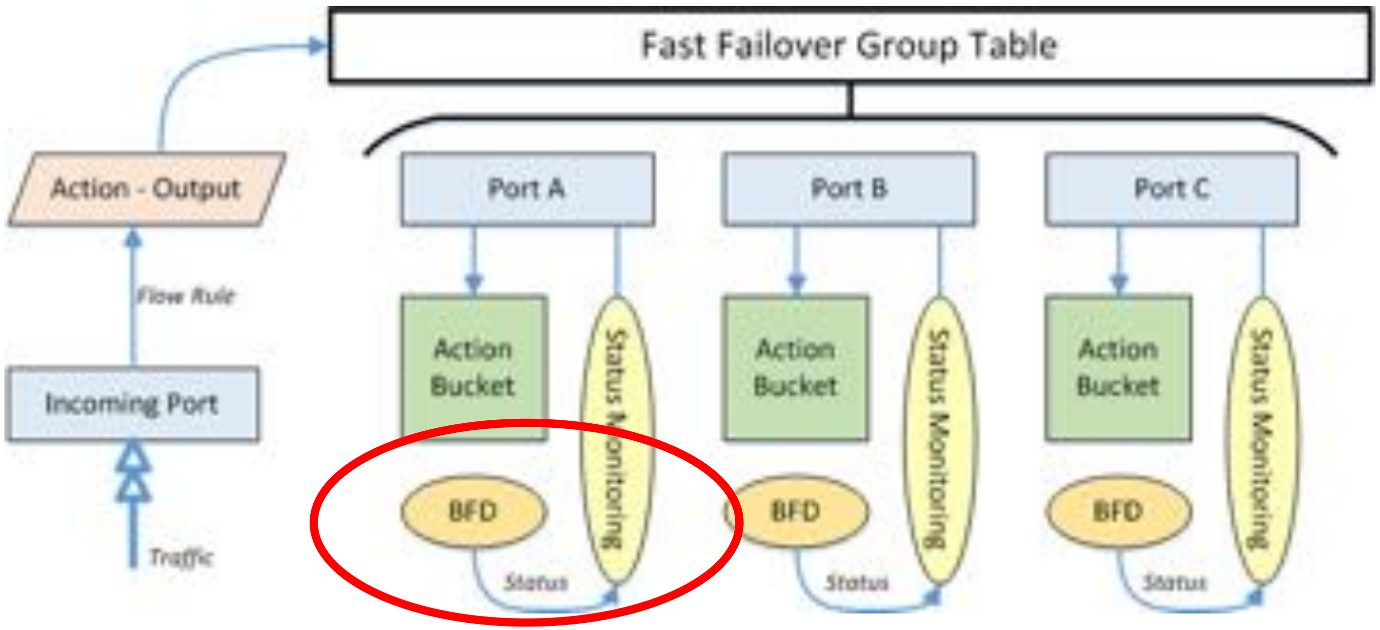
N. van Adrichem, C. Doerr, and F.A. Kuipers, “OpenNetMon: Network Monitoring in OpenFlow Software-Defined Networks,” Proc. of IEEE/IFIP NOMS, 2014.

Failures are bound to happen



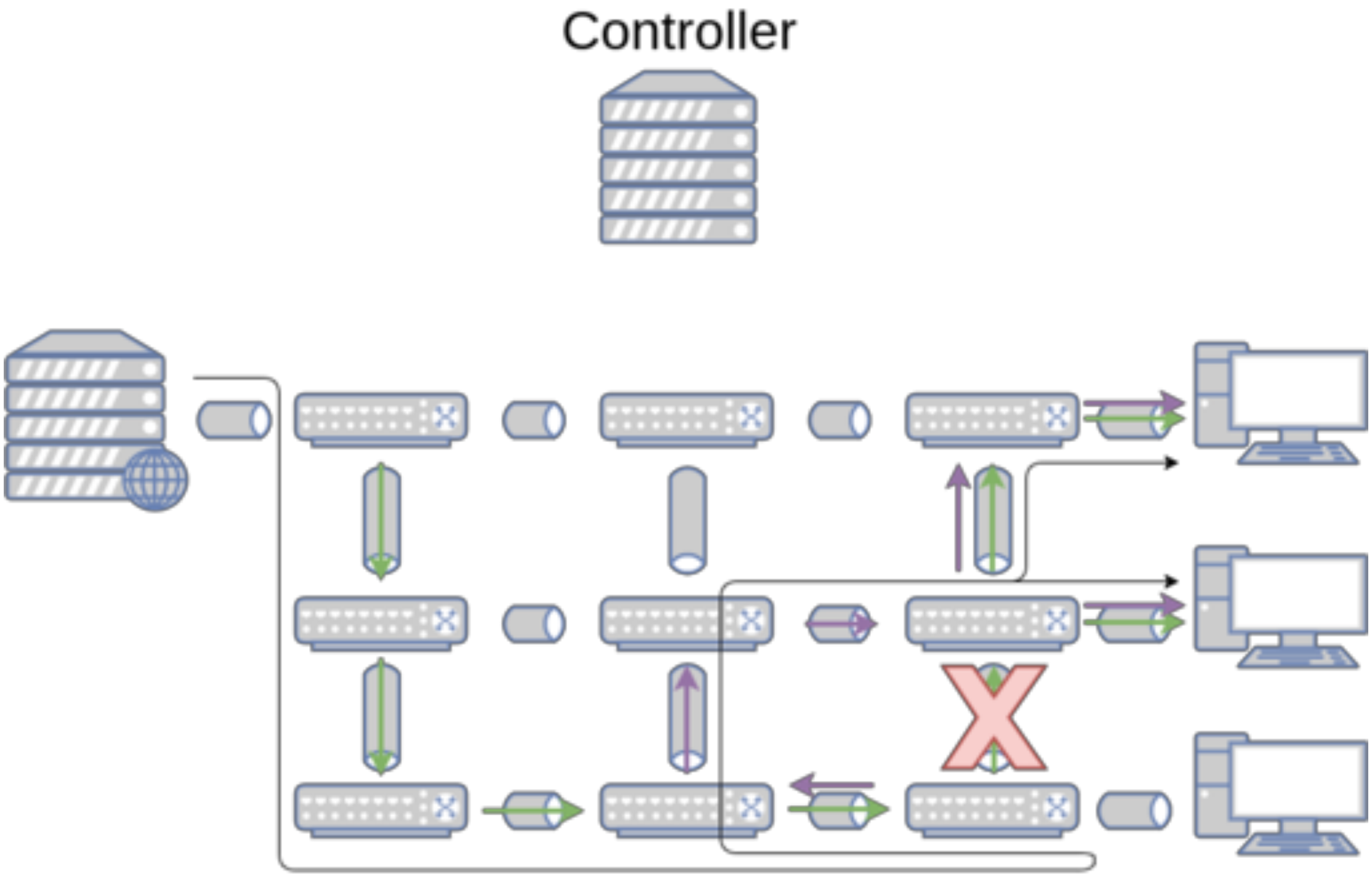
Waiting for the controller to install new rules is too time consuming!

Fast Failover



1. Fast recovery: Pre-install backup paths
2. Fast detection: Couple per-link BFD to Fast Failover buckets
   The switch executes the actions of the first bucket with a life watch port
3. Slow optimality: Rely on controller to reconfigure

- N. van Adrichem, B. van Asten, and F.A. Kuipers, "Fast Recovery in Software-Defined Networks, Proc. of EWSDN 2014.
- N.L.M. van Adrichem, F. Iqbal, and F.A. Kuipers, "Backup rules in Software-Defined Networks," Proc. of IEEE NFV-SDN 2016.



Purple: back-up entries

SD-WAN

WAN: A network spanning a large geographical area

B4: Experience with a Globally-Deployed Software Defined WAN

Sushant Jain, Alok Kumar, Subhasree Mandal, Joon Ong, Leon Poutievski, Arjun Singh, Subbaiah Venkata, Jim Wanderer, Junlan Zhou, Min Zhu, Jonathan Zolla, Urs Hölzle, Stephen Stuart and Amin Vahdat
Google, Inc.
b4-sigcomm@google.com

ABSTRACT

We present the design, implementation, and evaluation of B4, a private WAN connecting Google's data centers across the planet. B4 has a number of unique characteristics: i) massive bandwidth requirements deployed to a modest number of sites, ii) elastic traffic demand that seeks to maximize average bandwidth, and iii) full control over the edge servers and network, which enables rate limiting and demand measurement at the edge. These characteristics led to a Software Defined Networking architecture using OpenFlow to control relatively simple switches built from merchant silicon. B4's centralized traffic engineering service drives links to near 100% utilization, while splitting application flows among multiple paths to balance capacity against application priority/demands. We describe experience with three years of B4 production deployment, lessons learned, and areas for future work.

Categories and Subject Descriptors

C.2.2 [**Network Protocols**]: Routing Protocols

Keywords

Centralized Traffic Engineering; Wide-Area Networks; Software-Defined Networking; Routing; OpenFlow

1. INTRODUCTION

Modern wide area networks (WANs) are critical to Internet performance and reliability, delivering terabits/sec of aggregate bandwidth across thousands of individual links. Because individual WAN links are expensive and because WAN packet loss is typically thought unacceptable, WAN routers consist of high-end, specialized equipment that place a premium on high availability. Finally, WANs typically treat all bits the same. While this has many benefits, when the inevitable failure does take place, all applications are typically treated equally, despite their highly variable sensitivity to available capacity.

Given these considerations, WAN links are typically provisioned to 30-40% average utilization. This allows the network service provider to mask virtually all link or router failures from clients. Such overprovisioning delivers admirable reliability at the very real costs of 2-3x bandwidth over-provisioning and high-end routing gear.

We were faced with these overheads for building a WAN connecting multiple data centers with substantial bandwidth requirements. However, Google's data center WAN exhibits a number of unique characteristics. First, we control the applications, servers, and the LANs all the way to the edge of the network. Second, our most bandwidth-intensive applications perform large-scale data copies from one site to another. These applications benefit most from high levels of average bandwidth and can adapt their transmission rate based on available capacity. They could similarly defer to higher priority interactive applications during periods of failure or resource constraint. Third, we anticipated no more than a few dozen data center deployments, making central control of bandwidth feasible.

We exploited these properties to adopt a software defined networking (SDN) architecture for our data center WAN interconnect. We were most motivated by deploying routing and traffic engineering protocols customized to our unique requirements. Our design centers around: i) accepting failures as inevitable and common events, whose effects should be exposed to end applications, and ii) switch hardware that exports a simple interface to program forwarding table entries under central control. Network protocols could then run on servers housing a variety of standard and custom protocols. Our hope was that deploying novel routing, scheduling, monitoring, and management functionality and protocols would be both simpler and result in a more efficient network.

We present our experience deploying Google's WAN, B4, using Software Defined Networking (SDN) principles and OpenFlow [31] to manage individual switches. In particular, we discuss how we simultaneously support standard routing protocols and centralized Traffic Engineering (TE) as our first SDN application. With TE, we: i) leverage control at our network edge to adjudicate among competing demands during resource constraint, ii) use multipath forwarding/tunneling to leverage available network capacity according to application priority, and iii) dynamically reallocate bandwidth in the face of link/switch failures or shifting application demands. These features allow many B4 links to run at near 100% utilization and all links to average 70% utilization over long time periods, corresponding to 2-3x efficiency improvements relative to standard practice.

B4 has been in deployment for three years, now carries more traffic than Google's public facing WAN, and has a higher growth rate. It is among the first and largest SDN/OpenFlow deployments. B4 scales to meet application bandwidth demands more efficiently than would otherwise be possible, supports rapid deployment and iteration of novel control functionality such as TE, and enables tight integration with end applications for adaptive behavior in response to failures or changing communication patterns. SDN is of course

SIGCOMM'13, August 12–16, 2013, Hong Kong, China.

IXP + SDN = SDX

SDX: A Software Defined Internet Exchange

Arpit Gupta†, Laurent Vanbever⋆, Muhammad Shahbaz†, Sean P. Donovan†, Brandon Schlinker‡
Nick Feamster†, Jennifer Rexford⋆, Scott Shenker◇, Russ Clark†, Ethan Katz-Bassett‡
†Georgia Tech ⋆Princeton University ◇UC Berkeley ‡Univ. of Southern California

Abstract

BGP severely constrains how networks can deliver traffic over the Internet. Today's networks can only forward traffic based on the destination IP prefix, by selecting among routes offered by their immediate neighbors. We believe Software Defined Networking (SDN) could revolutionize wide-area traffic delivery, by offering direct control over packet-processing rules that match on multiple header fields and perform a variety of actions. Internet exchange points (IXPs) are a compelling place to start, given their central role in interconnecting many networks and their growing importance in bringing popular content closer to end users.

To realize a Software Defined IXP (an "SDX"), we must create compelling applications, such as "application-specific peering"—where two networks peer only for (say) streaming video traffic. We also need new programming abstractions that allow participating networks to create and run these applications and a runtime that both behaves correctly when interacting with BGP and ensures that applications do not interfere with each other. Finally, we must ensure that the system scales, both in rule-table size and computational overhead. In this paper, we tackle these challenges and demonstrate the flexibility and scalability of our solutions through controlled and in-the-wild experiments. Our experiments demonstrate that our SDX implementation can implement representative policies for hundreds of participants who advertise full routing tables while achieving sub-second convergence in response to configuration changes and routing updates.

Categories and Subject Descriptors: C.2.1 [Computer-Communication Networks] *Network Architecture and Design*: Network Communications

General Terms: Algorithms, Design, Experimentation

Keywords: software defined networking (SDN); Internet exchange point (IXP); BGP

1 Introduction

Internet routing is unreliable, inflexible, and difficult to manage. Network operators must rely on arcane mechanisms to perform traffic engineering, prevent attacks, and realize peering agreements. Internet routing's problems result from three characteristics of the Border Gateway Protocol (BGP), the Internet's interdomain routing protocol:

- **Routing only on destination IP prefix.** BGP selects and exports routes for destination prefixes. Networks cannot make more fine-grained decisions based on the type of application or the sender.
- **Influence only over direct neighbors.** A network selects among BGP routes learned from its direct neighbors, and exports selected routes to these neighbors. Networks have little control over end-to-end paths.
- **Indirect expression of policy.** Networks rely on indirect, obscure mechanisms (*e.g.*, "local preference", "AS Path Prepending") to influence path selection. Networks cannot directly express preferred inbound and outbound paths.

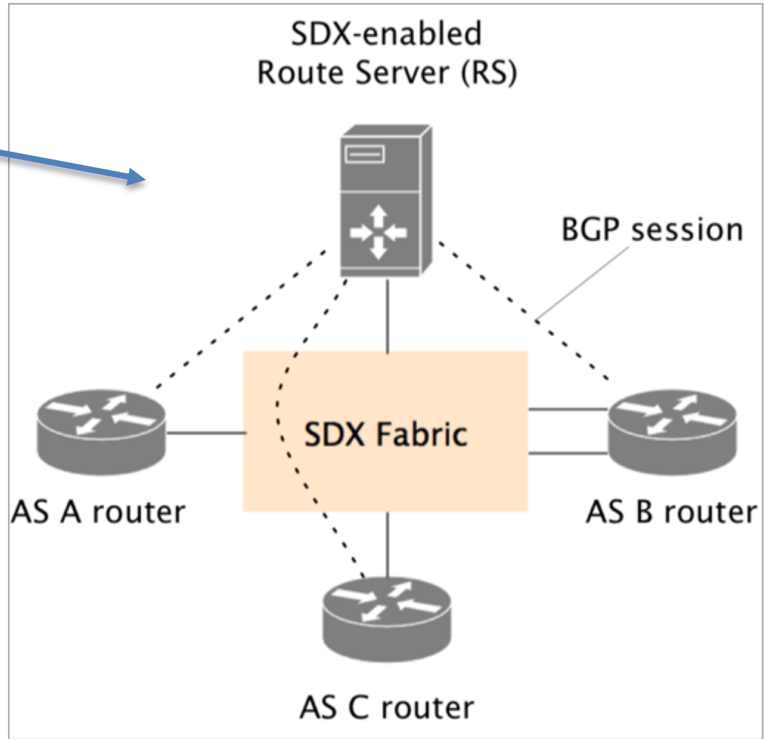
These problems are well-known, yet incremental deployment of alternative solutions is a perennial problem in a global Internet with more than 50,000 independently operated networks and a huge installed base of BGP-speaking routers.

In this paper, we develop a way forward that improves our existing routing system by allowing a network to execute a far wider range of decisions concerning end-to-end traffic delivery. Our approach builds on recent technology trends and also recognizes the need for incremental deployment. First, we believe that Software Defined Networking (SDN) shows great promise for simplifying network management and enabling new networked services. SDN switches match on a variety of header fields (not just destination prefix), perform a range of actions (not just forwarding), and offer direct control over the data plane. Yet, SDN currently only applies to *intra*domain settings, such as individual data-center, enterprise, or backbone networks. By design, a conventional SDN controller has purview over the switches within a single administrative (and trust) domain.

Second, we recognize the recent resurgence of interest in Internet exchange points (IXPs), which are physical locations where multiple networks meet to exchange traffic and BGP routes. An IXP is a layer-two network that, in the simplest case, consists of a single switch. Each participating network exchanges BGP routes (often with a BGP route server) and directs traffic to other participants over the layer-two fabric. The Internet has more than 300 IXPs worldwide—with more than 80 in North America alone—and some IXPs carry as much traffic as the tier-1 ISPs [1, 4]. For example, the Open IX effort seeks to develop new North American IXPs with open peering and governance, similar to the models already taking root in Europe. As video traffic continues to increase, tensions grow between content providers and access networks, and IXPs are on the front line of today's peering disputes. In short, not only are IXPs the right place to begin a revolution in wide-area traffic delivery, but the organizations running these IXPs have strong incentives to innovate.

We aim to change wide-area traffic delivery by designing, prototyping, and deploying a software defined exchange (SDX). Contrary to how it may seem, however, merely operating SDN switches and a controller at an IXP does not automatically present a turnkey solution. SDN is merely a tool for solving problems, not the solution.

SIGCOMM'14, August 17–22, 2014, Chicago, IL, USA.
Copyright 2014 ACM 978-1-4503-2836-4/14/08 ...$15.00.
http://dx.doi.org/10.1145/2619239.2626300

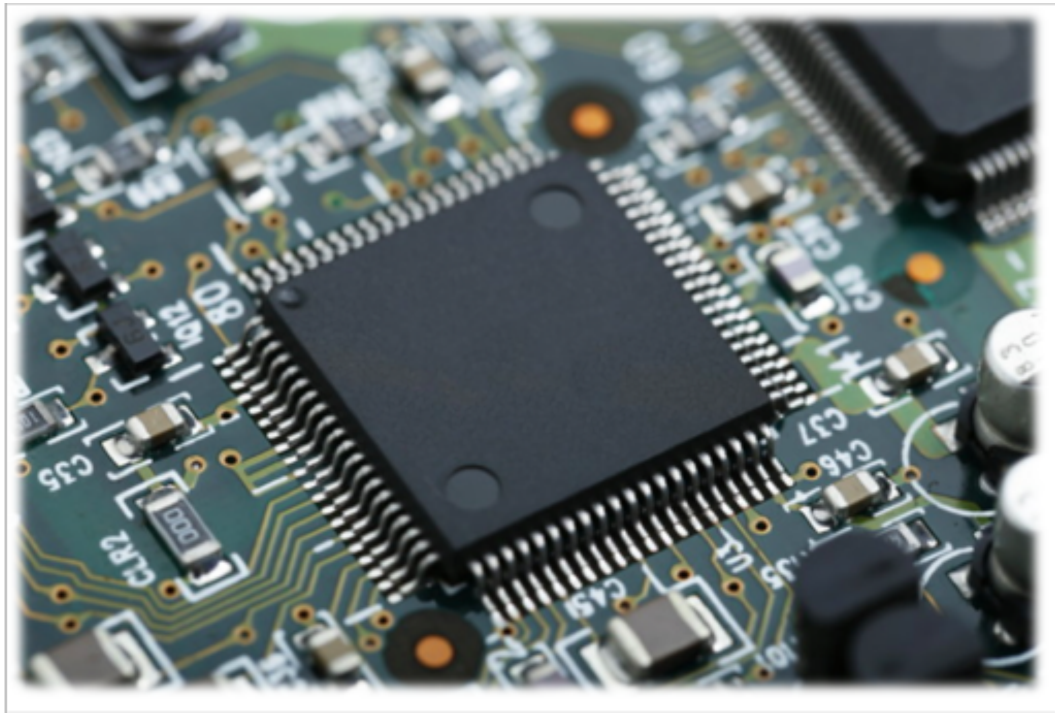


Network programmability

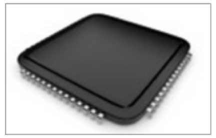
- SDN controller: programmable control plane
- What about the data plane?

Source: https://commons.wikimedia.org/wiki/File:Cisco-rs1.jpg

The limitation was in the hardware...



Fixed function ASIC



User programmable device!

coded in

TUDelft

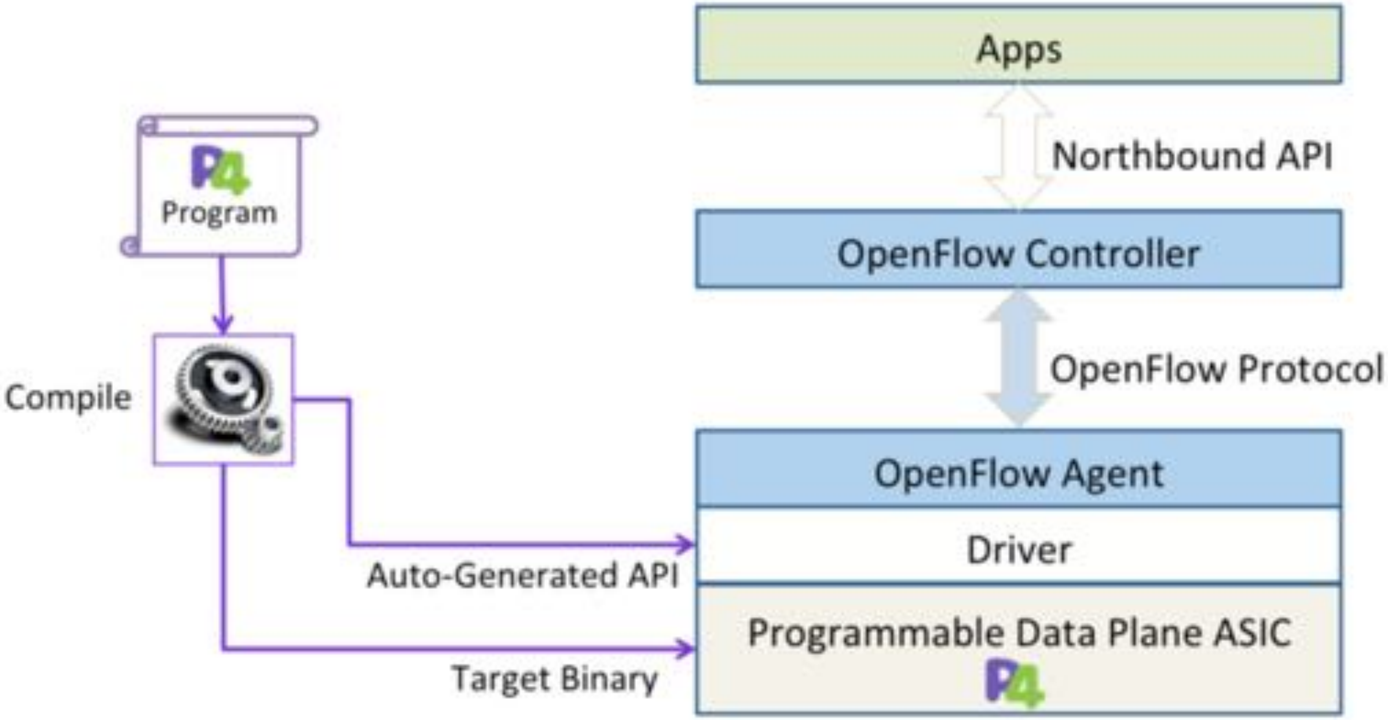
OpenFlow limitations

For every new protocol new headers need to be added to the OF specification and implemented by switch vendors

Version	Date	#Headers
OF 1.0	Dec 2009	12
OF 1.1	Feb 2011	15
OF 1.2	Dec 2011	36
OF 1.3	Jun 2012	40
OF 1.4	Oct 2013	41
OF 1.5	Mar 2015	45

TUDelft

OpenFlow can be a P4 program!



Why data-plane programmability?

- The device behaves exacly as you want
- Easy to update functionality
- Remove unused features
- See inside the data plane:
  - Telemetry information (queue occupancy, latency, time-stamps) can be used while forwarding the packets

The complexity of a P4 program

```
#include <core.p4>
#include <v1model.p4>
struct metadata {}
struct headers {}

parser MyParser(packet_in packet,
   out headers hdr,
   inout metadata meta,
   inout standard_metadata_t standard_metadata) {

    state start { transition accept; }
}

control MyVerifyChecksum(inout headers hdr, inout
metadata meta)
{
    apply { }
}

control MyIngress(inout headers hdr,
   inout metadata meta,
   inout std_mtdt_t std_mtdt) {
apply {
        if (std_mtdt.ingress_port == 1) {
            std_mtdt.egress_spec = 2;
        } else if (std_mtdt.ingress_port == 2) {
            std_mtdt.egress_spec = 1;
        }
    }
}
```

```
control MyEgress(inout headers hdr,
   inout metadata meta,
   inout standard_metadata_t standard_metadata) {
    apply {  }
}

control MyComputeChecksum(inout headers hdr, inout
metadata meta) {
     apply { }
}

control MyDeparser(packet_out packet, in headers
hdr) {
    apply { }
}

V1Switch(
   MyParser(),
   MyVerifyChecksum(),
   MyIngress(),
   MyEgress(),
   MyComputeChecksum(),
   MyDeparser()
) main;
```

Intent-Based Networking needed

```
import drop_heavy_hitters
import drop_ddos

define intent dropHeavyHitters:
  to     any
  for    traffic('any')
  apply  drop_heavy_hitters
  with   threshold('more', 20)

define intent dropDDoS:
  to     any
  for    traffic('any')
  apply  drop_ddos
  with   threshold('more', 5)
```

Similar-to-English network intent language

Reproduced from P4.org, "P4 Language Tutorial", 2018.

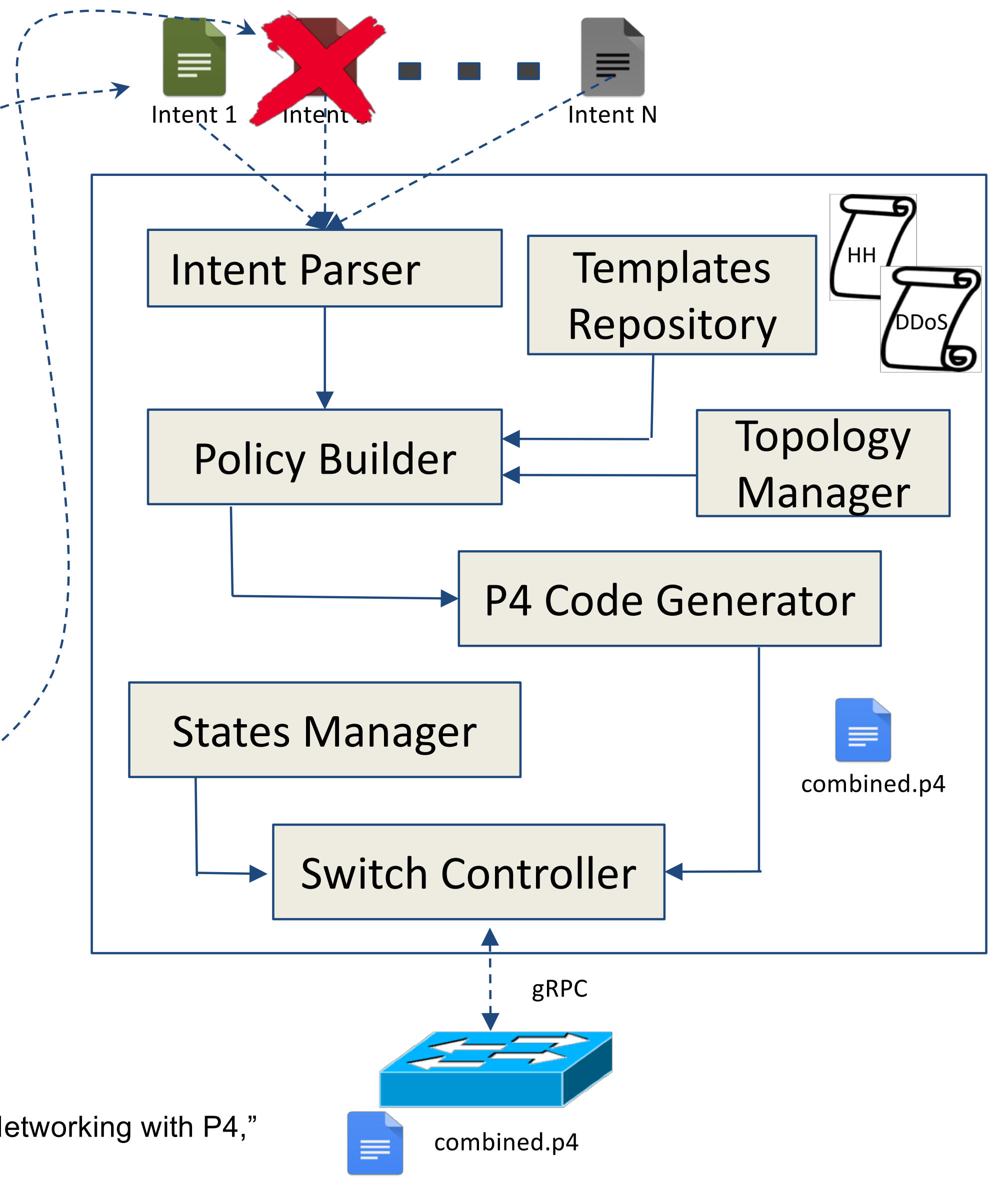
P4I/O

High-Level Diagram

```
import drop_heavy_hitters
import drop_ddos

define intent dropHeavyHitters:
  to     any
  for    traffic('any')
  apply  drop_heavy_hitters
  with   threshold('more', 20)

de      DDoS:
  to     any
  for    traffi
           _ddos
  with   threshold('more', 5)
...
```



M. Riftadi and F.A. Kuipers, “P4I/O: Intent-Based Networking with P4,” under submisison.

And why is this relevant for the Tactile Internet?

Low-latency challenge

- Congestion – when a network node is trying to forward more data than the outgoing link can process – causes queuing delay

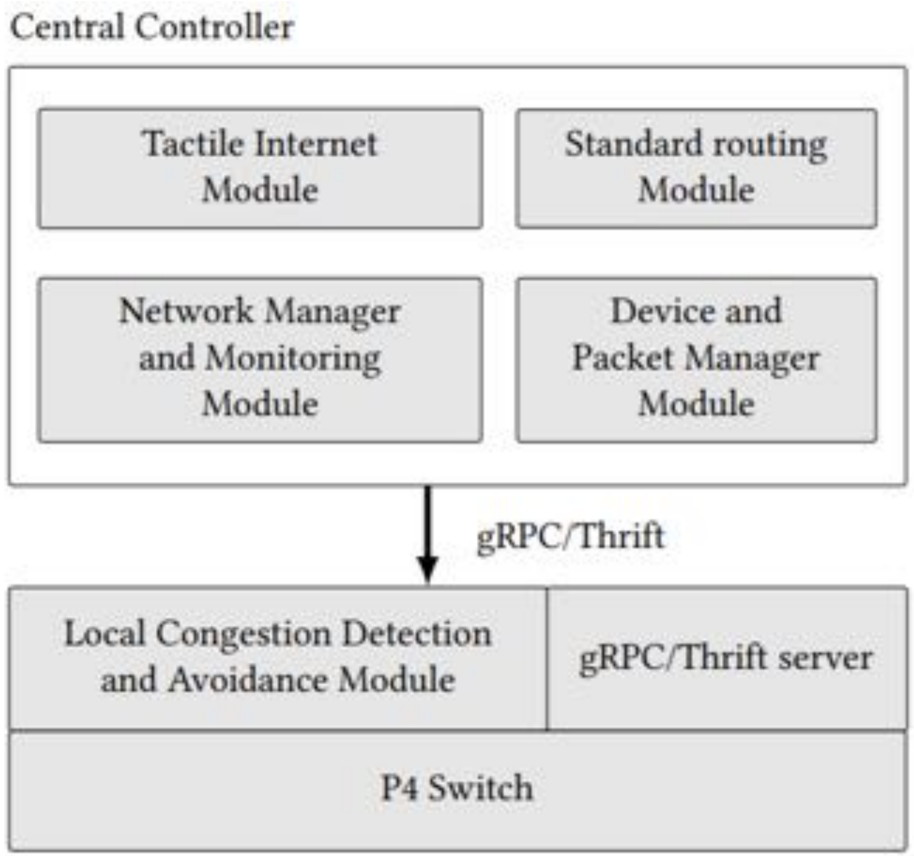
- Existing congestion control approaches are slow to react:
  - Transport layer reacts to RTT and/or loss
  - SDN is affected by controller-switch delay

TU Delft

Central + local control

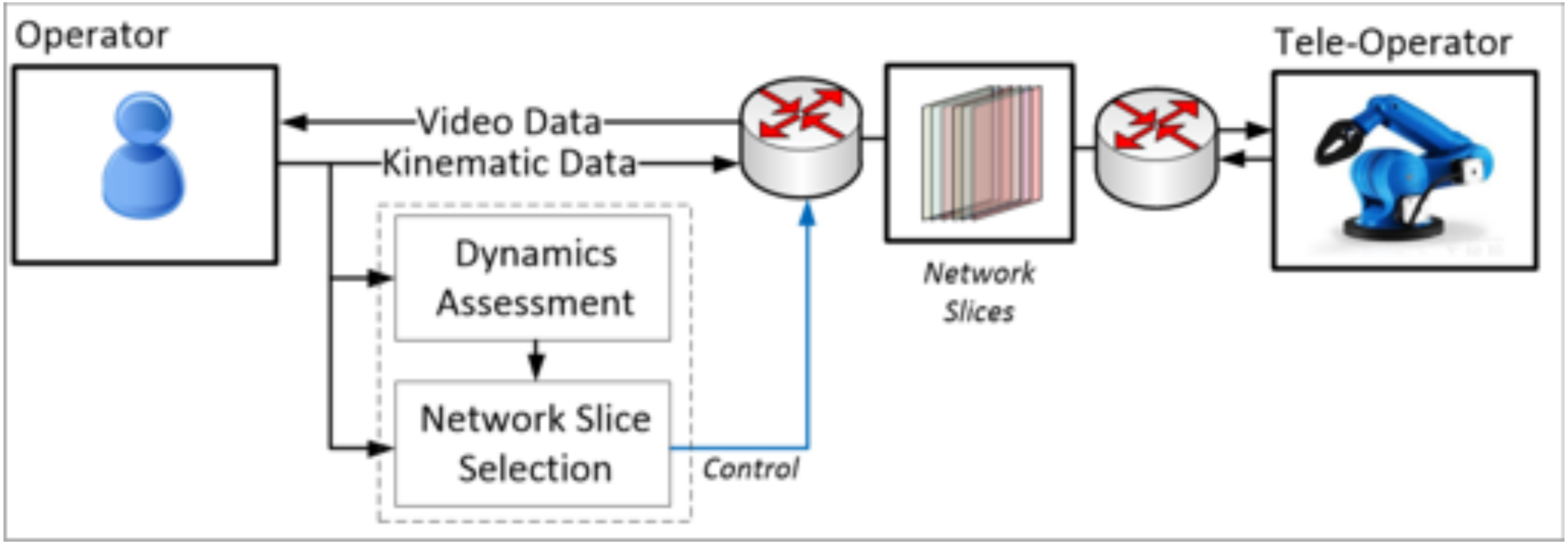
- **Central controller** – Configures and monitors paths for different service classes

- **Local controller** – Congestion detection and reaction in the data plane



B. Turkovic, F.A. Kuipers, N. van Adrichem, and K. Langendoen, “Fast network congestion detection and avoidance using P4”, Proc. of NEAT 2018.

Network slicing



That’s all folks!

Related work

1. K. Polachan, T.V. Prabhakar, C. Singh, and F.A. Kuipers, *Towards an Open Testbed for Tactile Cyber Physical Systems*, Proc. of COMSNETS 2019.
2. A. Indra Basuki and F.A. Kuipers, *Localizing link failures in legacy and SDN networks*, Proc. of RNDM 2018.
3. B. Turkovic, F.A. Kuipers, N. van Adrichem, and K. Langendoen, *Fast network congestion detection and avoidance using P4*, Proc. of NEAT 2018.
4. D. van den Berg, R. Glans, D. de Koning, F.A. Kuipers, J. Lugtenburg, K. Polachan, T.V. Prabhakar, C. Singh, B. Turkovic, and B. van Wijk, *Challenges in Haptic Communications over the Tactile Internet*, IEEE Access, Dec. 2017.
5. N. Bizanis and F.A. Kuipers, *SDN and virtualization solutions for the Internet of Things: A survey*, IEEE Access, Sep. 2016.
6. H. Krishna, N.L.M. van Adrichem, and F.A. Kuipers, *Providing Bandwidth Guarantees with OpenFlow*, Proc. of IEEE SCVT 2016.
7. N.L.M. van Adrichem, F. Iqbal, and F.A. Kuipers, *Backup rules in Software-Defined Networks*, Proc. of IEEE NFV-SDN 2016.
8. C. Mas Machuca, S. Secci, P. Vizarreta, F.A. Kuipers, A. Gouglidis, D. Hutchison, S. Jouet, D. Pezaros, A. Elmokashfi, P. Heegaard, and S. Ristov, *Technology-related Disasters: A Survey towards Disaster-resilient Software Defined Networks*, Proc. of RNDM 2016.
9. N. van Adrichem and F.A. Kuipers, *NDNFlow: Software-Defined Named Data Networking*, Proc. of IEEE NetSoft 2015.
10. M.P.V. Manthena, N. van Adrichem, C. van den Broek, and F.A. Kuipers, *An SDN-based Architecture for Network-as-a-Service*, Proc. of IEEE NetSoft 2015.
11. N. van Adrichem, B. van Asten, and F.A. Kuipers, *Fast Recovery in Software-Defined Networks*, Proc. of EWSDN 2014.
12. N. van Adrichem, C. Doerr, and F.A. Kuipers, *OpenNetMon: Network Monitoring in OpenFlow Software-Defined Networks*, Proc. of IEEE/IFIP NOMS 2014.
13. R. van der Pol, M. Bredel, A. Barczyk, B. Overeinder, N. van Adrichem, and F.A. Kuipers, *Experiences with MPTCP in an intercontinental OpenFlow network*, Proc. of TNC 2013.