wNetKAT: A Weighted SDN Programming and Verification Language

Kim G. Larsen, Stefan Schmid, Bingtian Xue

Aalborg University, DENMARK

OPODIS 2016 in Madrid, Spain

Computer Networks

- Computer networks (datacenter networks, enterprise networks, wide-area networks) have become a critical infrastructure of the information society

- Concern: are today's networks dependable and flexible enough?

Even techsavvy companies struggle to provide reliable operations

We discovered a misconfiguration on this pair of switches that caused what's called a "bridge loop" in the network.

A network change was […] executed incorrectly […] more "stuck" volumes and added more requests to the re-mirroring storm

Service outage was due to a series of internal network events that corrupted router data tables

Experienced a network connectivity issue […] interrupted the airline's flight departures, airport processing and reservations systems

Source: Talk by Nate Foster at DSDN Workshop

Another Anecdote: Wall-Street Bank

- Outage of a data center of a Wall Street investment bank
- Lost revenue measured in USD 10^6 / min!
- Quickly, an emergency team was assembled with experts in compute, storage and networking:
  - **The compute team**: came armed with reams of logs, showing how and when the applications failed, and had already written experiments to reproduce and isolate the error, along with candidate prototype programs to workaround the failure.
  - **The storage team**: similarly equipped, showing which file system logs were affected, and already progressing with workaround programs.
  - And the **networking team**? Only had ping and traceroute

Source: «The world's fastest and most programmable networks»
White Paper Barefoot Networks

Another Anecdote: Wall-Street Bank

"All the networking team had were two tools invented over twenty years ago to merely test end-to-end connectivity. Neither tool could reveal problems with the switches, the congestion experienced by individual packets, or provide any means to create experiments to identify, quarantine and resolve the problem. Whether or not the problem was in the network, the network team would be blamed since they were unable to demonstrate otherwise."

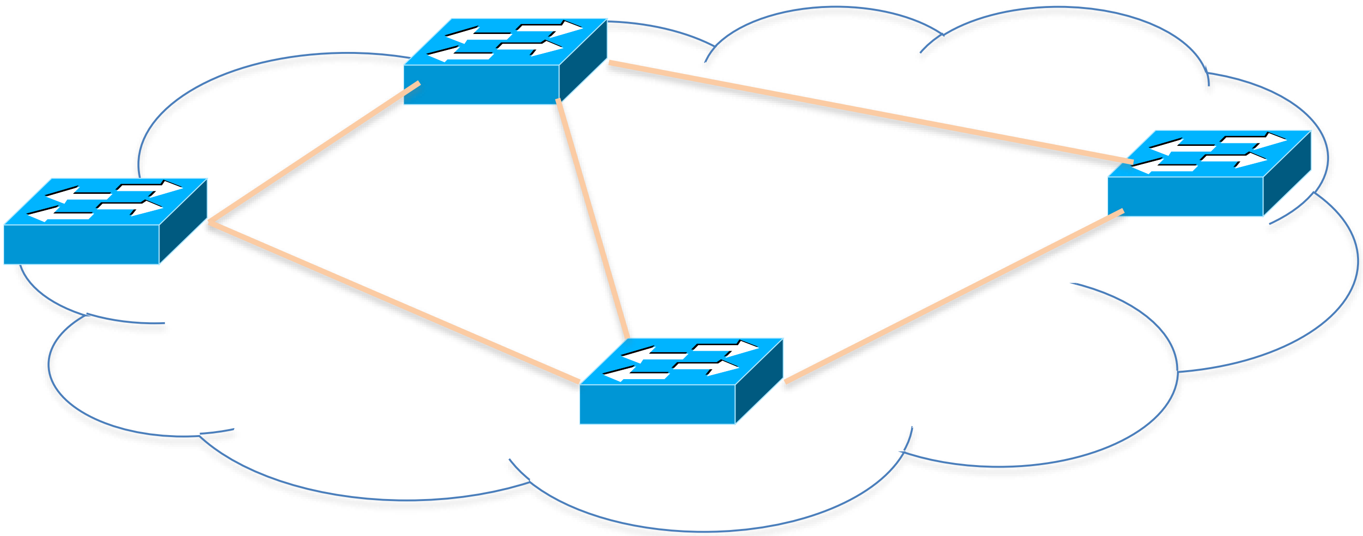
Software-Defined Networks (SDNs) promise to introduce networking innovations, by decoupling the control plane from the data plane, and by making networks programmable and verifiable automatically.

Source: «The world's fastest and most programmable networks»
White Paper Barefoot Networks

Traditional Networks: Data and Control Plane

Data plane:
Packet streaming

- Forward
- Filter
- Buffer
- Mark
- Rate-limit
- Measure packets

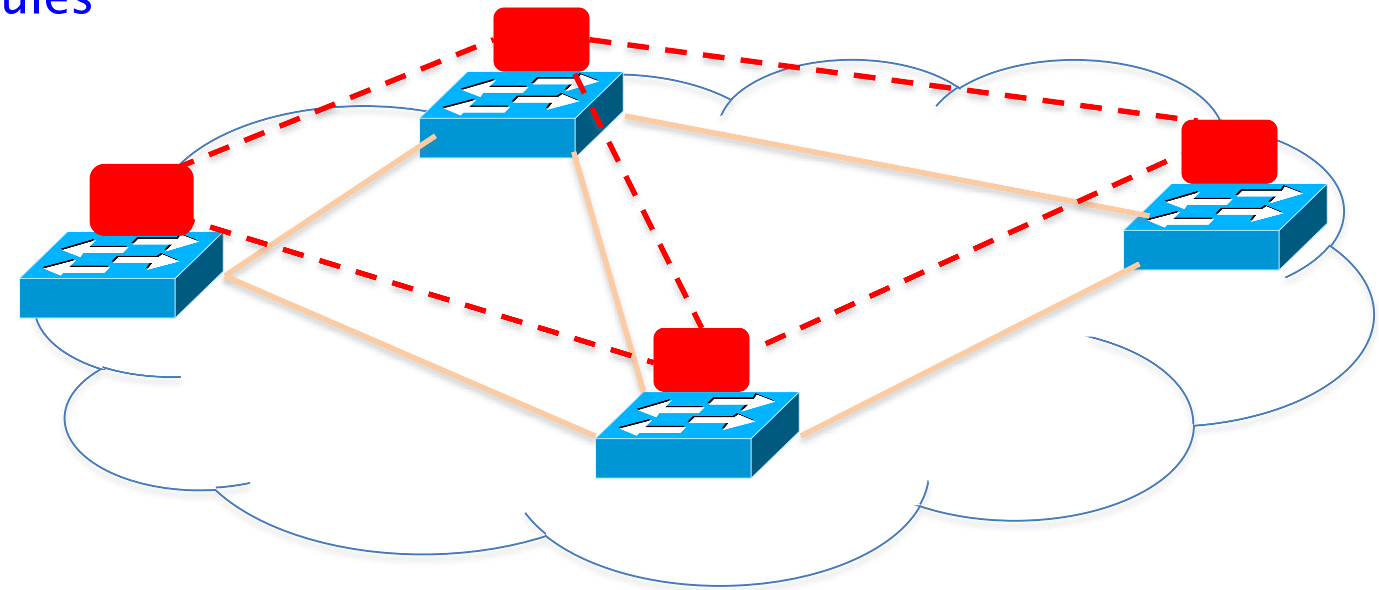


Traditional Networks: Data and Control Plane

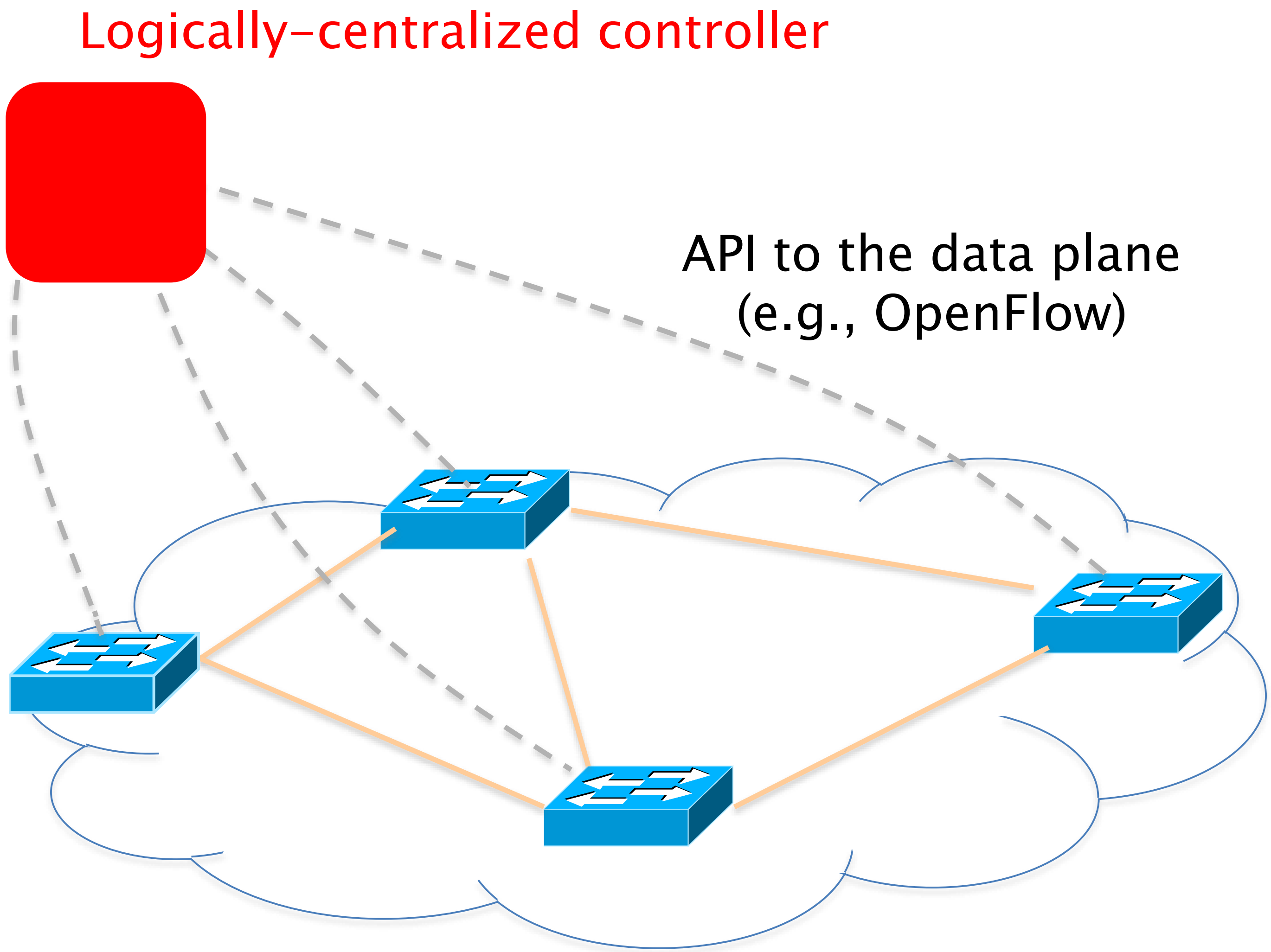
Control plane: Distributed

Distributed algorithms

Track topology changes
Compute routes
Install forwarding rules

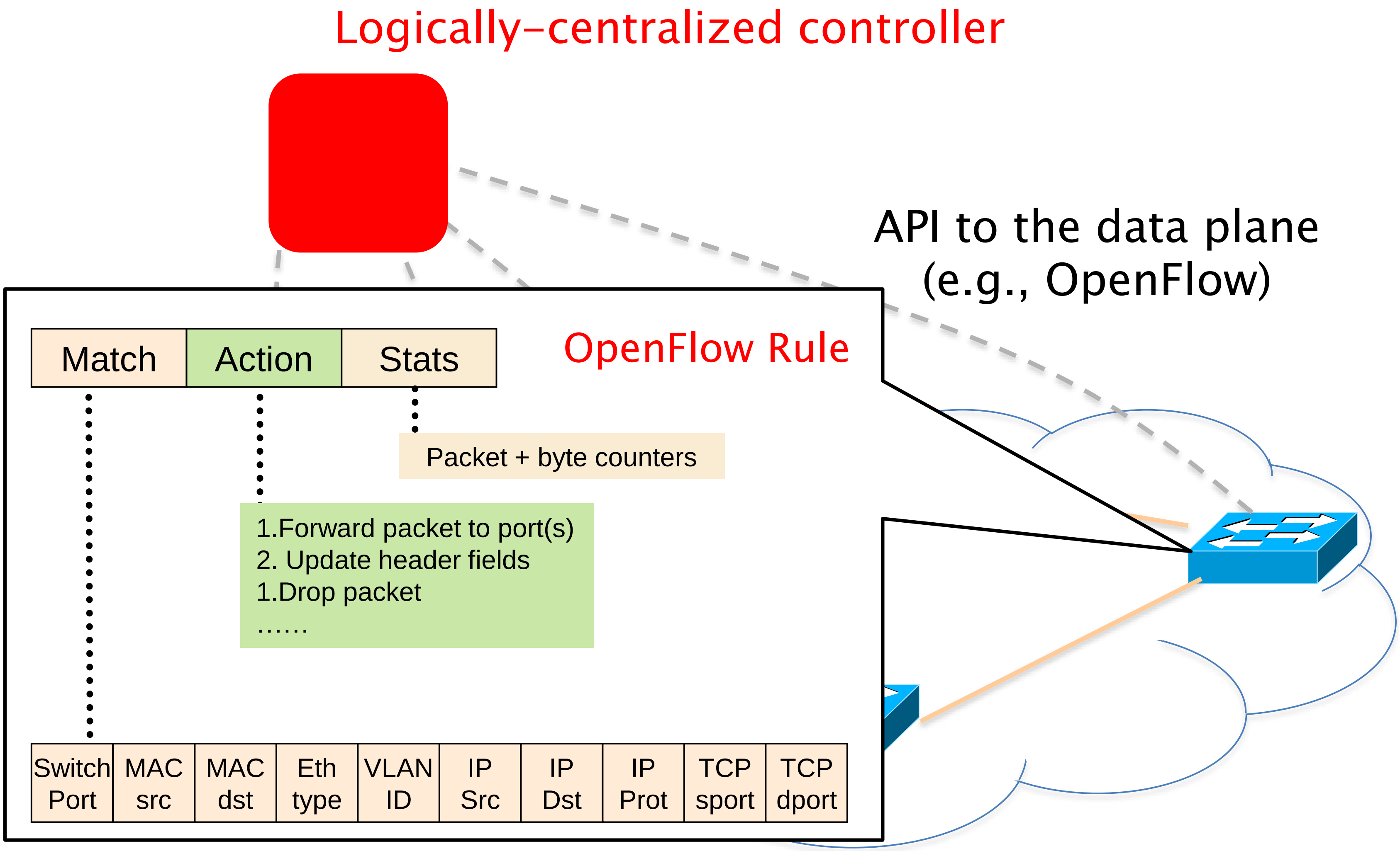


Software Defined Networks (SDN)

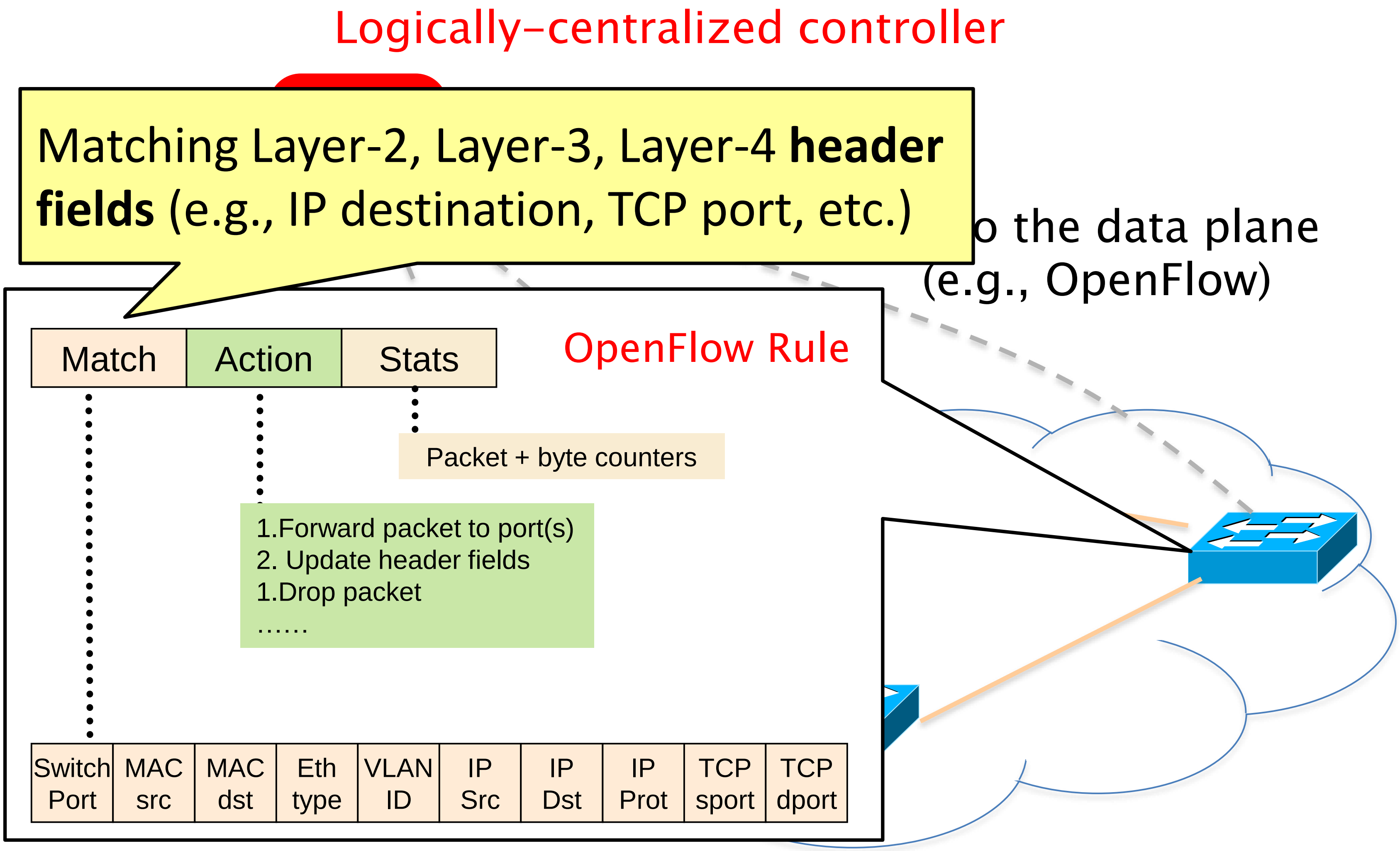


AALBORG UNIVERSITY

SDN/OpenFlow: Match-Action Devices

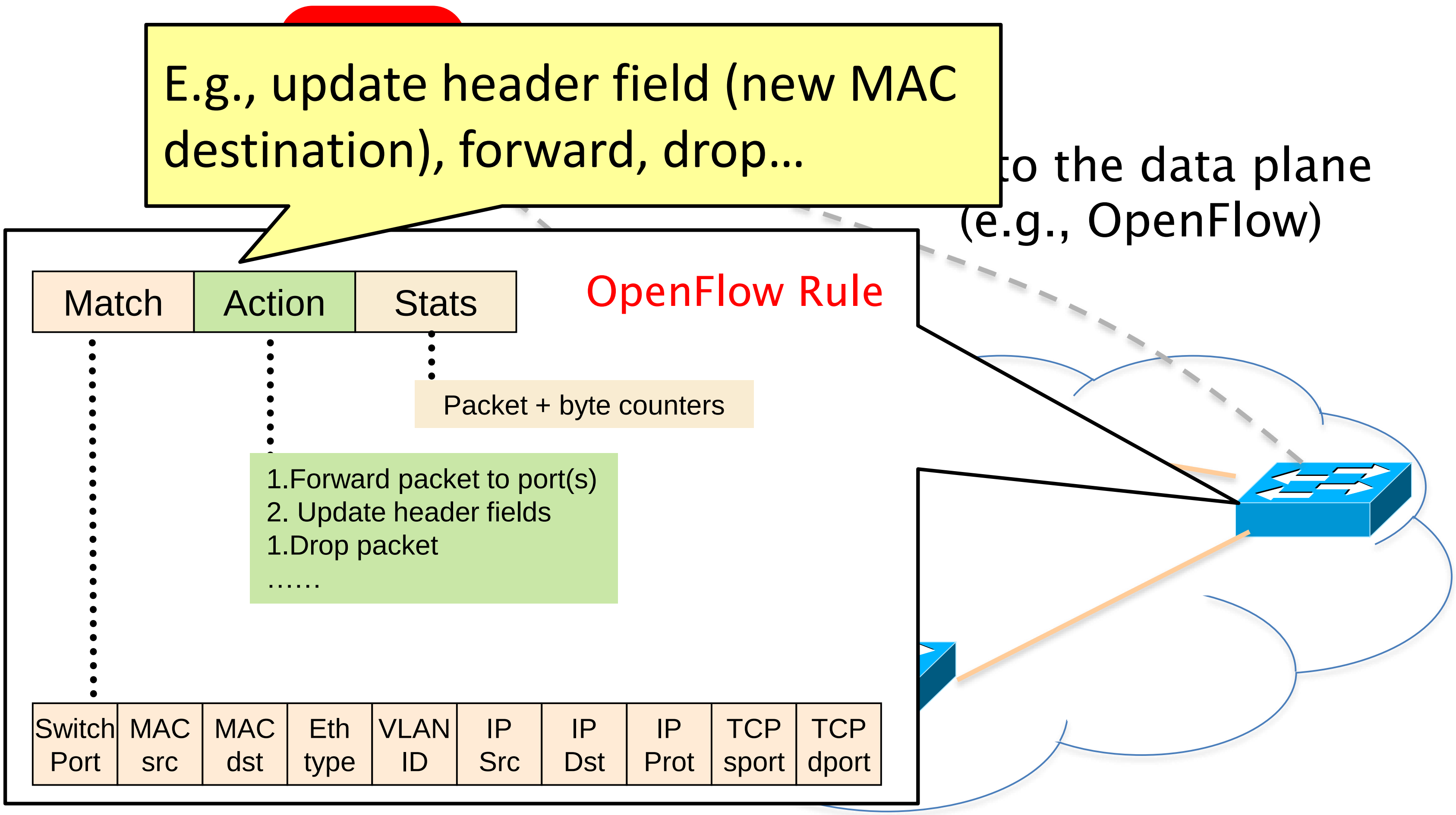


SDN/OpenFlow: Match-Action Devices

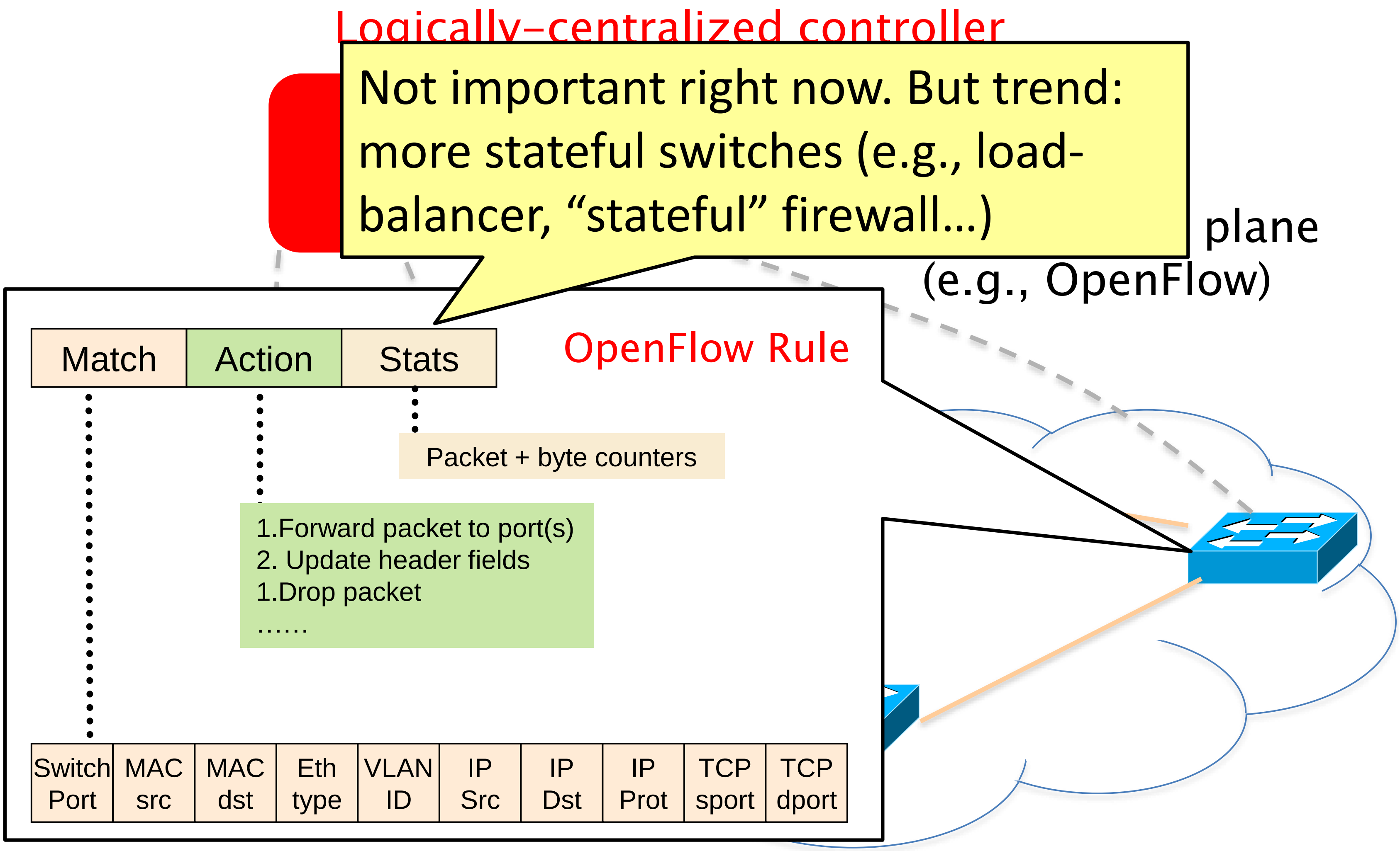


SDN/OpenFlow: Match-Action Devices

Logically-centralized controller



SDN/OpenFlow: Match-Action Devices

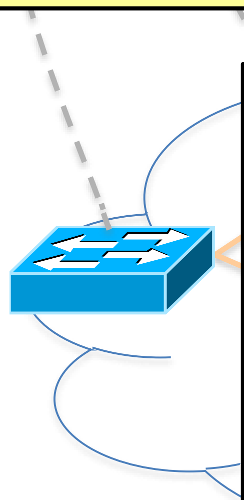


AALBORG UNIVERSITY

Benefits of SDN

<u>Benefit 1:</u> Decoupling! Control plane can evolve independently of data plane: innovation at speed of software development. And simpler network management through logically centralized view: many network management and operational tasks are inherently non-local.

plane
ow)



<u>Benefit 2:</u> Simple match-action devices: supports formal verification! Switches/ routers are “simple” and “passive” match-action devices. (Unlike in, e.g., active networks.) Can do, e.g., header space analysis.

Solution: *Software-Defined Networks* (SDN)!

- SDNs are popular: deployments in enterprises, datacenters, WAN, IXPs
- E.g., Google: "A Purpose-Built Global Network: Google's Move to SDN"

A Purpose-built Global Network: Google's Move to SDN

case study

A DISCUSSION WITH AMIN VAHDAT, DAVID CLARK, AND JENNIFER REXFORD



Everything about Google is at scale, of course—a market cap of legendary proportions, an unrivaled talent pool, enough intellectual property to keep armies of attorneys in Guccis for life, and, oh yeah, a private WAN (wide area network) bigger than you can possibly imagine that also happens to be growing substantially faster than the Internet as a whole.

Unfortunately, bigger isn't always better, at least not where networks are concerned, since along with massive size come massive costs, bigger management challenges, and the knowledge that traditional solutions probably aren't going to cut it. And then there's this: specialized network gear doesn't come cheap.

Adding it all up, Google found itself on a cost curve it considered unsustainable. Perhaps even worse, it saw itself at the mercy of a small number of network equipment vendors that have proved to be slow in terms of delivering the capabilities requested by the company. Which is why Google

Programming SDN

- SDN is about programming the networks
- But OpenFlow is very low-level: inconvenient for programmers
- Hence, over the last years, many network specific programming languages have been developed
- Researchers have started developing more high-level languages
- NetKAT: state-of-the-art framework for programming and reasoning about networks

NetKAT

NetKAT: Kleene Algebra with Tests (KAT) with atoms like:

- $f \leftarrow w$ assignment
- $f = w$ test

Assigning values to header fields

Testing values in header fields

NetKAT: No need to understand all details now ☺

Syntax

Fields	$f ::=$	$f_1 \mid \cdots \mid f_k$	
Packets	$pk ::=$	$\{f_1 = v_1, \cdots, f_k = v_k\}$	
Histories	$h ::=$	$pk::\langle\rangle \mid pk::h$	
Predicates	$a, b ::=$	1	*Identity*
	$\mid$	0	*Drop*
	$\mid$	$f = n$	*Test*
	$\mid$	$a + b$	*Disjunction*
	$\mid$	$a \cdot b$	*Conjunction*
	$\mid$	$\neg a$	*Negation*
Policies	$p, q ::=$	a	*Filter*
	$\mid$	$f \leftarrow n$	*Modification*
	$\mid$	$p + q$	*Union*
	$\mid$	$p \cdot q$	*Sequential composition*
	$\mid$	p^*	*Kleene star*
	$\mid$	dup	*Duplication*

Semantics

$$[\![p]\!] \in \mathsf{H} \to \mathcal{P}(\mathsf{H})$$

$$[\![1]\!]\ h \triangleq \{h\}$$

$$[\![0]\!]\ h \triangleq \{\}$$

$$[\![f = n]\!]\ (pk::h) \triangleq \begin{cases} \{pk::h\} & \text{if } pk.f = n \\ \{\} & \text{otherwise} \end{cases}$$

$$[\![\neg a]\!]\ h \triangleq \{h\} \setminus ([\![a]\!]\ h)$$

$$[\![f \leftarrow n]\!]\ (pk::h) \triangleq \{pk[f := n]::h\}$$

$$[\![p + q]\!]\ h \triangleq [\![p]\!]\ h \cup [\![q]\!]\ h$$

$$[\![p \cdot q]\!]\ h \triangleq ([\![p]\!] \bullet [\![q]\!])\ h$$

$$[\![p^*]\!]\ h \triangleq \bigcup_{i \in \mathbb{N}} F^i\ h$$

$$\text{where } F^0\ h \triangleq \{h\} \text{ and } F^{i+1}\ h \triangleq ([\![p]\!] \bullet F^i)\ h$$

$$[\![\mathsf{dup}]\!]\ (pk::h) \triangleq \{pk::(pk::h)\}$$

NetKAT … stand all details now ☺

Important concept 1:
Packets with a set of fields

Syntax

Fields	$f ::=$	$f_1 \mid \cdots \mid f_k$	
Packets	$pk ::=$	$\{f_1 = v_1, \cdots, f_k = v_k\}$	
Histories	$h ::=$	$pk::\langle\rangle \mid pk::h$	
Predicates	$a, b ::=$	1	*Identity*
	$\mid$	0	*Drop*
	$\mid$	$f = n$	*Test*
	$\mid$	$a + b$	*Disjunction*
	$\mid$	$a \cdot b$	*Conjunction*
	$\mid$	$\neg a$	*Negation*
Policies	$p, q ::=$	a	*Filter*
	$\mid$	$f \leftarrow n$	*Modification*
	$\mid$	$p + q$	*Union*
	$\mid$	$p \cdot q$	*Sequential composition*
	$\mid$	p^*	*Kleene star*
	$\mid$	dup	*Duplication*

Semantics

$$[\![p]\!] \in \mathsf{H} \rightarrow \mathcal{P}(\mathsf{H})$$

$$[\![1]\!]\ h \triangleq \{h\}$$

$$[\![0]\!]\ h \triangleq \{\}$$

$$[\![f = n]\!]\ (pk::h) \triangleq \begin{cases} \{pk::h\} & \text{if } pk.f = n \\ \{\} & \text{otherwise} \end{cases}$$

$$[\![\neg a]\!]\ h \triangleq \{h\} \setminus ([\![a]\!]\ h)$$

$$[\![f \leftarrow n]\!]\ (pk::h) \triangleq \{pk[f := n]::h\}$$

$$[\![p + q]\!]\ h \triangleq [\![p]\!]\ h \cup [\![q]\!]\ h$$

$$[\![p \cdot q]\!]\ h \triangleq ([\![p]\!] \bullet [\![q]\!])\ h$$

$$[\![p^*]\!]\ h \triangleq \bigcup_{i \in \mathbb{N}} F^i\ h$$

$$\text{where } F^0\ h \triangleq \{h\} \text{ and } F^{i+1}\ h \triangleq ([\![p]\!] \bullet F^i)\ h$$

$$[\![\mathsf{dup}]\!]\ (pk::h) \triangleq \{pk::(pk::h)\}$$

NetKAT: No need to understand all details now ☺

Important concept 2: **history**. We maintain packet history that records the state of each packet as it travels from switch to switch.

Synta

Fields	$f ::=$		
Packets	$pk ::=$	$\ldots = v_1, \cdots, f_k = v_k\}$	
Histories	$h ::=$	$pk::\langle\rangle \mid pk::h$	
Predicates	$a, b ::=$	1	*Identity*
	$\mid$	0	*Drop*
	$\mid$	$f = n$	*Test*
	$\mid$	$a + b$	*Disjunction*
	$\mid$	$a \cdot b$	*Conjunction*
	$\mid$	$\neg a$	*Negation*
Policies	$p, q ::=$	a	*Filter*
	$\mid$	$f \leftarrow n$	*Modification*
	$\mid$	$p + q$	*Union*
	$\mid$	$p \cdot q$	*Sequential composition*
	$\mid$	p^*	*Kleene star*
	$\mid$	dup	*Duplication*

$$[\![p]\!] \in \mathsf{H} \rightarrow \mathcal{P}(\mathsf{H})$$

$$[\![1]\!]\ h \triangleq \{h\}$$

$$[\![0]\!]\ h \triangleq \{\}$$

$$[\![f = n]\!]\ (pk::h) \triangleq \begin{cases} \{pk::h\} & \text{if } pk.f = n \\ \{\} & \text{otherwise} \end{cases}$$

$$[\![\neg a]\!]\ h \triangleq \{h\} \setminus ([\![a]\!]\ h)$$

$$[\![f \leftarrow n]\!]\ (pk::h) \triangleq \{pk[f := n]::h\}$$

$$[\![p + q]\!]\ h \triangleq [\![p]\!]\ h \cup [\![q]\!]\ h$$

$$[\![p \cdot q]\!]\ h \triangleq ([\![p]\!] \bullet [\![q]\!])\ h$$

$$[\![p^*]\!]\ h \triangleq \bigcup_{i \in \mathbb{N}} F^i\ h$$

where $F^0\ h \triangleq \{h\}$ and $F^{i+1}\ h \triangleq ([\![p]\!] \bullet F^i)\ h$

$$[\![\mathsf{dup}]\!]\ (pk::h) \triangleq \{pk::(pk::h)\}$$

NetKAT: No need to understand all details now ☺

Syntax

Fields	$f ::=$	$f_1 \mid \cdots \mid f_k$	
Packets	$pk ::=$	$\{f_1 = v_1, \cdots, f_k = v_k\}$	
Histories	$h ::=$	$pk::\langle\rangle \mid pk::h$	
Pr…			*Identity*
			…rop
			…est
			…isjunction
			…onjunction
			Negation
Policies	$p, q ::=$	a	*Filter*
	$\mid$	$f \leftarrow n$	*Modification*
	$\mid$	$p + q$	*Union*
	$\mid$	$p \cdot q$	*Sequential composition*
	$\mid$	p^*	*Kleene star*
	$\mid$	dup	*Duplication*

Policies includes, e.g., modification of header field.

Semantics

$$[\![p]\!] \in \mathsf{H} \rightarrow \mathcal{P}(\mathsf{H})$$

$$[\![1]\!]\ h \triangleq \{h\}$$

$$[\![0]\!]\ h \triangleq \{\}$$

$$[\![f = n]\!]\ (pk::h) \triangleq \begin{cases} \{pk::h\} & \text{if } pk.f = n \\ \{\} & \text{otherwise} \end{cases}$$

$$[\![\neg a]\!]\ h \triangleq \{h\} \setminus ([\![a]\!]\ h)$$

$$[\![f \leftarrow n]\!]\ (pk::h) \triangleq \{pk[f := n]::h\}$$

$$[\![p + q]\!]\ h \triangleq [\![p]\!]\ h \cup [\![q]\!]\ h$$

where F^0 …) h

[dup] …

We always work on the current packet, at front of history. (History: just to keep track of trajectory.)

NetKAT: No need to understand all details now ☺

Syntax

Fields $f ::= f_1 \mid \cdots \mid f_k$

$\ldots = v_k\}$

Predicates includes, e.g., test of header field.

Predicates $a, b ::=$	1	*Identity*
$\mid$	0	*Drop*
$\mid$	$f = n$	*Test*
$\mid$	$a + b$	*Disjunction*
$\mid$	$a \cdot b$	*Conjunction*
$\mid$	$\neg a$	*Negation*
Policies $p, q ::=$	a	*Filter*
$\mid$	$f \leftarrow n$	*Modification*
$\mid$	$p + q$	*Union*
$\mid$	$p \cdot q$	*Sequential composition*
$\mid$	p^*	*Kleene star*
$\mid$	dup	*Duplication*

Semantics

$$[\![p]\!] \in \mathsf{H} \rightarrow \mathcal{P}(\mathsf{H})$$

$$[\![1]\!]\ h \triangleq \{h\}$$

$$[\![0]\!]\ h \triangleq \{\}$$

$$[\![f = n]\!]\ (pk{::}h) \triangleq \begin{cases} \{pk{::}h\} & \text{if } pk.f = n \\ \{\} & \text{otherwise} \end{cases}$$

$$[\![\neg a]\!]\ h \triangleq \{h\} \setminus ([\![a]\!]\ h)$$

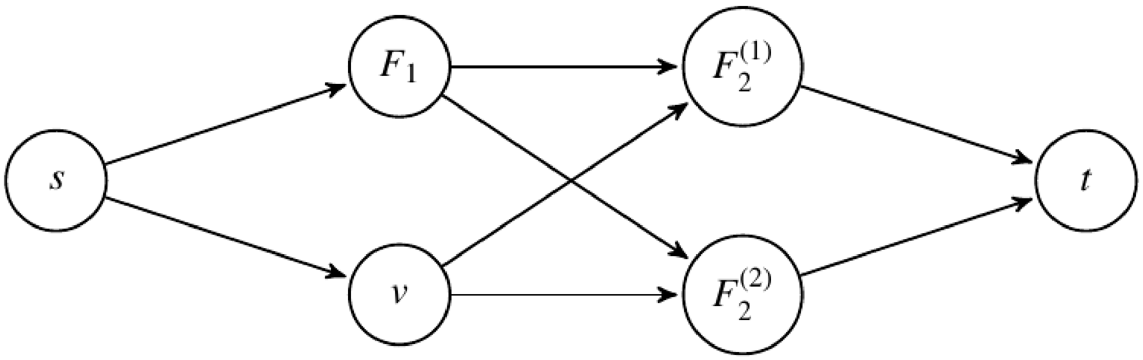
Producing an empty history = dropping the packet.
Producing a singleton = forwarding to a single port

$$\ldots \bullet F^i)\ h$$

$$[\![\mathsf{dup}]\!]\ (pk{::}h) \triangleq \{pk{::}(pk{::}h)\}$$

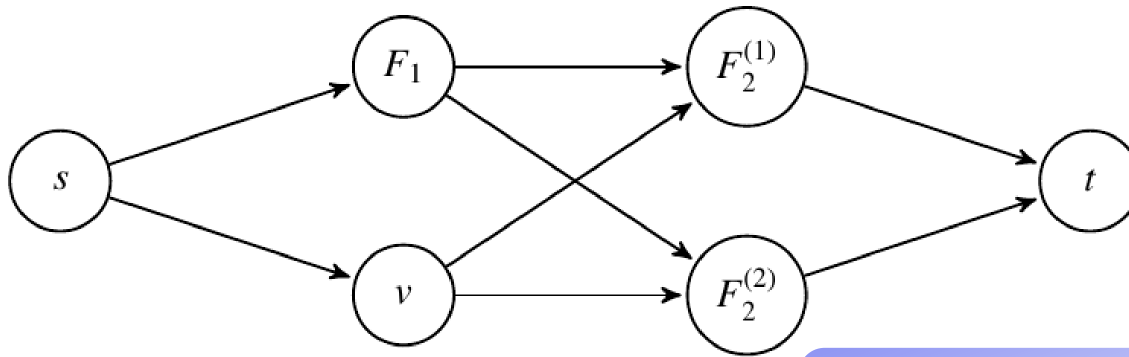
NetKAT: Example

Can model the topology as the union of smaller policies that encode the behavior of each link.



NetKAT: Example

Can model the topology as the union of smaller policies that encode the behavior of each link.



possible modifications

$$
\begin{aligned}
t \quad ::= \quad & sw = s; (sw \leftarrow F_1 + sw \leftarrow v) \\
& + sw = F_1; (sw \leftarrow F_2^{(1)} + sw \leftarrow F_2^{(2)}) \\
& + sw = v; (sw \leftarrow F_1^{(1)} + sw \leftarrow F_2^{(2)}) \\
& + sw = F_2^{(1)}; sw \leftarrow t \\
& + sw = F_2^{(2)}; sw \leftarrow t
\end{aligned}
$$

union

NetKAT: Use Cases

NetKAT allows to answer many important questions:

- "Can X connect to Y?"
- "Is traffic from A to B routed through Z?"
- "Is there a loop involving S?"
- "Are non-SSH packets forwarded?"
- Etc.

NetKAT: Use Cases

NetKAT allows to answer many important questions:

- "Can X connect to Y?"
- "Is traffic from A to B routed through Z?"
- "Is there a loop involving S?"
- "Are non-SSH packets forwarded?"
- Etc.

However, NetKAT is limited to binary contexts. What is missing today is a framework to reason about the inherent weighted aspects of networking: **wNetKAT**.

wNetKAT: Example

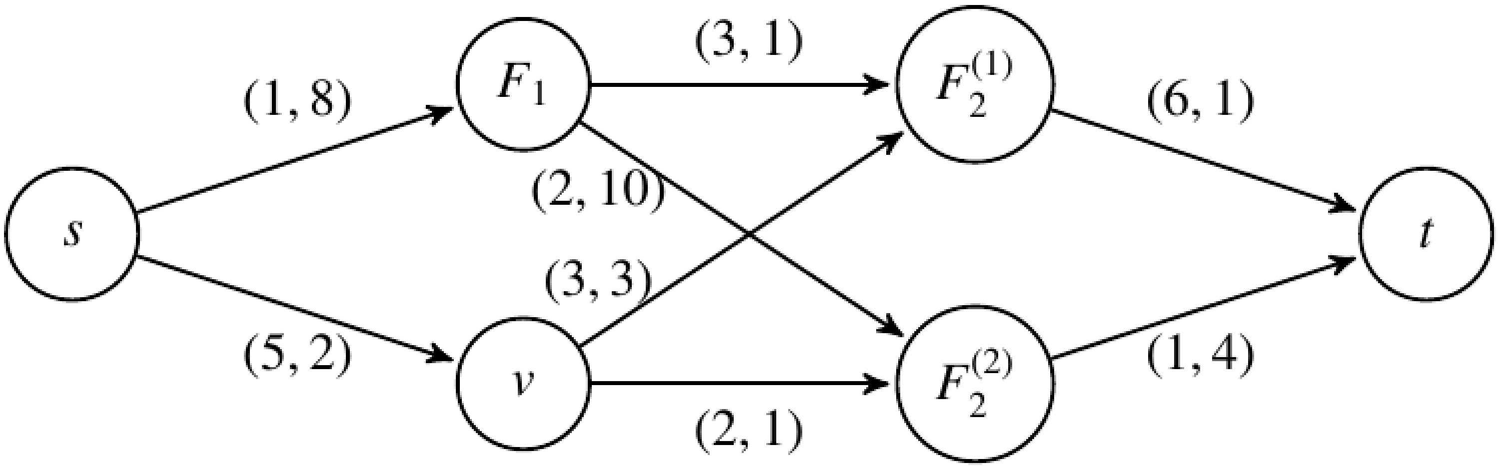
Real networks are weighted: links have costs (latency, energy, peering costs, etc.) and are capacitated (e.g., bandwidth).

But also nodes may have capacity constraints or entail costs. Moreover, nodes may transform the traffic volume (e.g., add or remove encapsulation headers or compress packets).

wNetKAT: Example

Real networks are weighted: links have costs (latency, energy, peering costs, etc.) and are capacitated (e.g., bandwidth).

But also nodes may have capacity constraints or entail costs. Moreover, nodes may transform the traffic volume (e.g., add or remove encapsulation headers or compress packets).



The Case for Weighted NetKAT!

wNetKAT: Challenges

The weighted extension of NetKAT is non-trivial:

- capacity constraints introduce dependencies between flows (e.g., packets compete for bandwidth)
- we need arithmetic operations such as addition (e.g., in case of latency to compute the end-to-end delay) or minimum (e.g., in case of bandwidth)

Therefore, we extend the syntax of NetKAT toward weighted packet- and switch-variables, as well as queues, and provide a semantics accordingly.

Paper Contributions

- We show for which weighted aspects and use cases which language extensions are required.
- We show the relation between WNetKAT expressions and weighted finite automata: an important operational model for weighted programs.
  - This also leads to the undeciability of WNetKAT equivalence problem.
- We explore the complexity of verification more generally and for subsets of the language
  - We prove the decidability of whether an expression equals 0 (emptiness testing): for many practical scenarios a sufficient and relevant problem (e.g., reachability)

wNetKAT: Additions to NetKAT

We add two types of variables to NetKAT:

- quantitative packet variables!
- (quantitative, non-quantitative) switch variables

Accordingly, we generalize assignment and test, to also include arithmetic operations (namely addition):

- Quantitative Assignment
- Quantitative Test

wNetKAT: Additions to NetKAT

Also allows us to model more stateful switches (as they are currently underway)

We add two types of variables to NetK

- quantitative packet variables!
- (quantitative, non-quantitative) switch variables

Accordingly, we generalize assignment and test, to also include arithmetic operations (namely addition):

- Quantitative Assignment
- Quantitative Test

wNetKAT: Another no-go slide! ☺

$$[\![x \leftarrow \omega]\!](\rho,\ pk :: h) = \begin{cases} \{\rho,\ pk[\omega/x] :: h\} & \text{if } x \in \mathcal{V}_p \\ \{\rho(v)[\omega/x],\ pk :: h\} & \text{if } x \in \mathcal{V}_s \text{ and } pk(sw) = v \end{cases}$$

$$[\![x = \omega]\!](\rho,\ pk :: h) = \begin{cases} \{\rho,\ pk :: h\} & \text{if } x \in \mathcal{V}_p \text{ and } pk(x) = \omega \\ & \text{ or if } x \in \mathcal{V}_s, pk(sw) = v \text{ and } \rho(v, x) = \omega \\ \emptyset & \text{otherwise} \end{cases}$$

$$[\![y \leftarrow (\Sigma_{y' \in \mathcal{V}'} y' + r)]\!](\rho,\ pk :: h) = \begin{cases} \{\rho,\ pk[r'/x] :: h\} & \text{if } x \in \mathcal{V}_p \\ \{\rho(v)[r'/x],\ pk :: h\} & \text{if } x \in \mathcal{V}_s \text{ and } pk(sw) = v \end{cases}$$

where $r' = \Sigma_{y_p \in \mathcal{V}' \cap \mathcal{V}_p} pk(y_p) + \Sigma_{y_s \in \mathcal{V}' \cap \mathcal{V}_q} \rho(v, y_s) + r$

$$[\![y = (\Sigma_{y' \in \mathcal{V}'} y' + r)]\!](\rho,\ pk :: h) = \begin{cases} \{\rho,\ pk :: h\} & \text{if } x \in \mathcal{V}_p \text{ and } pk(x) = r' \\ & \text{ or } x \in \mathcal{V}_s, pk(sw) = v \text{ and } \rho(v, x) = r' \\ \emptyset & \text{otherwise} \end{cases}$$

where $r' = \Sigma_{y_p \in \mathcal{V}' \cap \mathcal{V}_p} pk(y_p) + \Sigma_{y_s \in \mathcal{V}' \cap \mathcal{V}_q} \rho(v, y_s) + r$

$$x \in \mathcal{V}_n, y \in \mathcal{V}_q$$

wNetKAT: Another no-go slide! ☺

Quantitative update: update the corresponding header field if x is a packet-variable, or update the corresponding switch information of the current switch if x is a switch-variable..

$[\![x$ … $= \omega$

… otherwise

$$[\![y \leftarrow (\Sigma_{y' \in \mathcal{V}'} y' + r)]\!](\rho,\ pk :: h) = \begin{cases} \{\rho,\ pk[r'/x] :: h\} & \text{if } x \in \mathcal{V}_p \\ \{\rho(v)[r'/x],\ pk :: h\} & \text{if } x \in \mathcal{V}_s \text{ and } pk(sw) = v \end{cases}$$

$$\text{where } r' = \Sigma_{y_p \in \mathcal{V}' \cap \mathcal{V}_p} pk(y_p) + \Sigma_{y_s \in \mathcal{V}' \cap \mathcal{V}_q} \rho(v, y_s) + r$$

$$[\![y = (\Sigma_{y' \in \mathcal{V}'} y' + r)]\!](\rho,\ pk :: h) = \begin{cases} \{\rho,\ pk :: h\} & \text{if } x \in \mathcal{V}_p \text{ and } pk(x) = r' \\ & \text{or } x \in \mathcal{V}_s, pk(sw) = v \text{ and } \rho(v, x) = r' \\ \emptyset & \text{otherwise} \end{cases}$$

$$\text{where } r' = \Sigma_{y_p \in \mathcal{V}' \cap \mathcal{V}_p} pk(y_p) + \Sigma_{y_s \in \mathcal{V}' \cap \mathcal{V}_q} \rho(v, y_s) + r$$

$$x \in \mathcal{V}_n, y \in \mathcal{V}_q$$

wNetKAT: Another no-go slide! ☺

$$[\![x \leftarrow \omega]\!](\rho,\ pk :: h) = \begin{cases} \{\rho,\ pk[\omega/x] :: h\} & \text{if } x \in \mathcal{V}_p \\ \{\rho(v)[\omega/x],\ pk :: h\} & \text{if } x \in \mathcal{V}_s \text{ and } pk(sw) = v \end{cases}$$

$$[\![x = \omega]\!](\rho,\ pk :: h) = \begin{cases} \{\rho,\ pk :: h\} & \text{if } x \in \mathcal{V}_p \text{ and } pk(x) = \omega \\ & \text{ or if } x \in \mathcal{V}_s, pk(sw) = v \text{ and } \rho(v, x) = \omega \\ \emptyset & \text{otherwise} \end{cases}$$

$$[\![y \leftarrow (\Sigma_{y' \in \mathcal{V}'} y' + r)]\!](\rho,\ pk :: h) = \begin{cases} \{\rho,\ pk[r'/x] :: h\} & \text{if } x \in \mathcal{V}_p \\ \{\rho(v)[r'/x],\ pk :: h\} & \text{if } x \in \mathcal{V}_s \text{ and } pk(sw) = v \end{cases}$$

where $r' = \Sigma_{y_p \in \mathcal{V}' \cap \mathcal{V}_p} pk(y_p) + \Sigma_{y_s \in \mathcal{V}' \cap \mathcal{V}_q} \rho(v, y_s) + r$

$$[\![y = (\Sigma_{y' \in \mathcal{V}'} y' + r)]\!](\rho,\ pk :: h) = \begin{cases} \{\rho,\ pk :: h\} & \text{if } x \in \mathcal{V}_p \text{ and } pk(x) = r' \\ & \text{ or } x \in \mathcal{V}_s, pk(sw) = v \text{ and } \rho(v, x) = r' \\ \emptyset & \text{otherwise} \end{cases}$$

where $r' = \Sigma_{y_p \in \mathcal{V}' \cap \mathcal{V}_p} pk(y_p) + \Sigma_{y_s \in \mathcal{V}' \cap \mathcal{V}_q} \rho(v, y_s) + r$

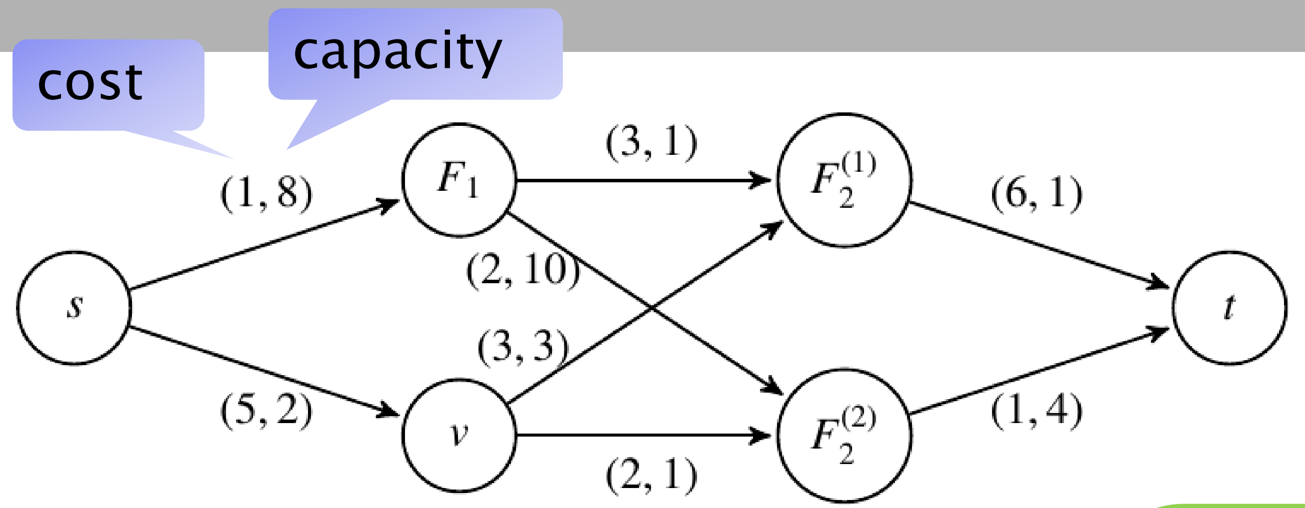
Test the quantitative variables using the current packet- and switch-variables.

wNetKAT

- We only support addition
- But we can also support other arithmetic operations
  - min or max can be easily defined (see paper):

$$x \leftarrow \min\{y, z\} \stackrel{\text{def}}{=} y \leq z; x \leftarrow y \& y > z; x \leftarrow z.$$

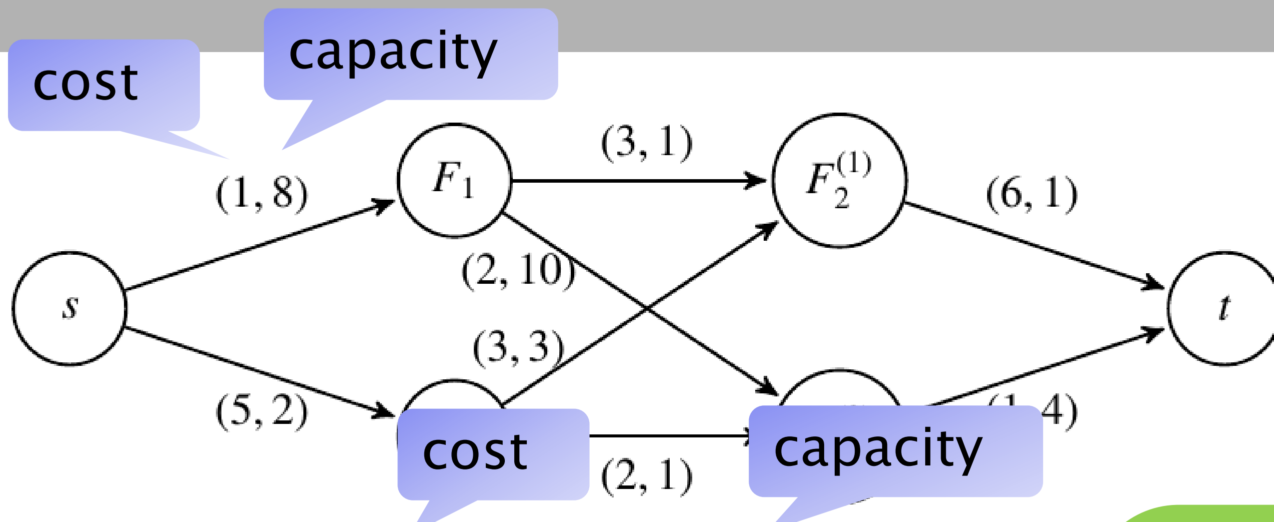
Example: Characterizing Weighted Topologies



$$
\begin{aligned}
t ::= \quad & sw = s; (sw \leftarrow F_1; co \leftarrow co + 1; ca \leftarrow \min\{ca, 8\} \\
& \qquad \& \; sw \leftarrow v; co \leftarrow co + 5; ca \leftarrow \min\{ca, 2\}) \\
& \& \; sw = F_1; \\
& \quad (sw \leftarrow F_2^{(1)}; co \leftarrow co + 3; ca \leftarrow \min\{ca, 1\} \\
& \quad \& \; sw \leftarrow F_2^{(2)}; co \leftarrow co + 2; ca \leftarrow \min\{ca, 10\}) \\
& \& \; sw = v; (sw \leftarrow F_2^{(1)}; co \leftarrow co + 3; ca \leftarrow \min\{ca, 3\} \\
& \qquad \& \; sw \leftarrow F_2^{(2)}; co \leftarrow co + 2; ca \leftarrow \min\{ca, 1\}) \\
& \& \; sw = F_2^{(1)}; sw \leftarrow t; co \leftarrow co + 6; ca \leftarrow \min\{ca, 1\} \\
& \& \; sw = F_2^{(2)}; sw \leftarrow t; co \leftarrow co + 1; ca \leftarrow \min\{ca, 4\}
\end{aligned}
$$

Can model the topology as the union of smaller policies that encode the behavior of each link.

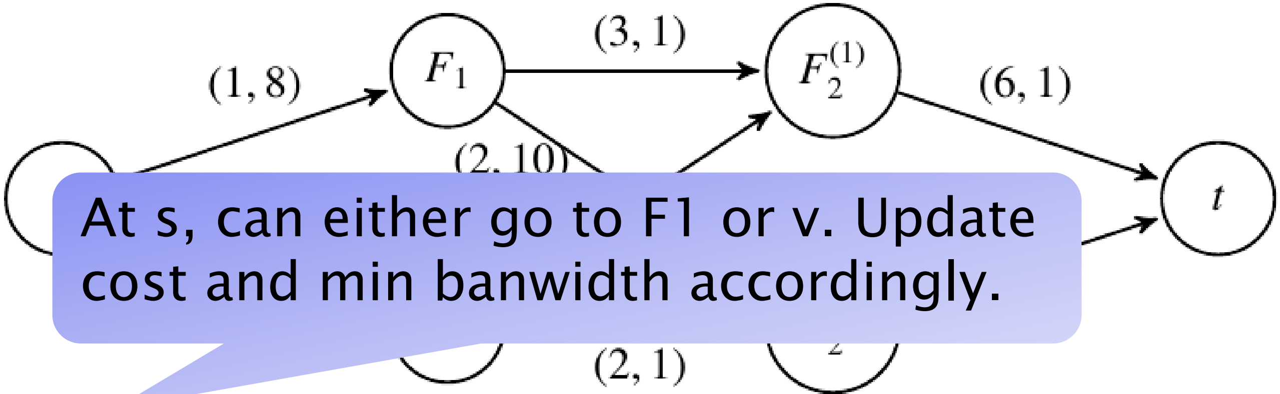
Example: Characterizing Weighted Topologies



$$
\begin{aligned}
t ::= \quad & sw = s; (sw \leftarrow F_1; co \leftarrow co + 1; ca \leftarrow \min\{ca, 8\} \\
& \qquad \& \ sw \leftarrow v; co \leftarrow co + 5; ca \leftarrow \min\{ca, 2\}) \\
& \& \ sw = F_1; \\
& \qquad (sw \leftarrow F_2^{(1)}; co \leftarrow co + 3; ca \leftarrow \min\{ca, 1\} \\
& \qquad \& \ sw \leftarrow F_2^{(2)}; co \leftarrow co + 2; ca \leftarrow \min\{ca, 10\}) \\
& \& \ sw = v; (sw \leftarrow F_2^{(1)}; co \leftarrow co + 3; ca \leftarrow \min\{ca, 3\} \\
& \qquad \& \ sw \leftarrow F_2^{(2)}; co \leftarrow co + 2; ca \leftarrow \min\{ca, 1\}) \\
& \& \ sw = F_2^{(1)}; sw \leftarrow t; co \leftarrow co + 6; ca \leftarrow \min\{ca, 1\} \\
& \& \ sw = F_2^{(2)}; sw \leftarrow t; co \leftarrow co + 1; ca \leftarrow \min\{ca, 4\}
\end{aligned}
$$

Can model the topology as the union of smaller policies that encode the behavior of each link.

Example: Characterizing Weighted Topologies



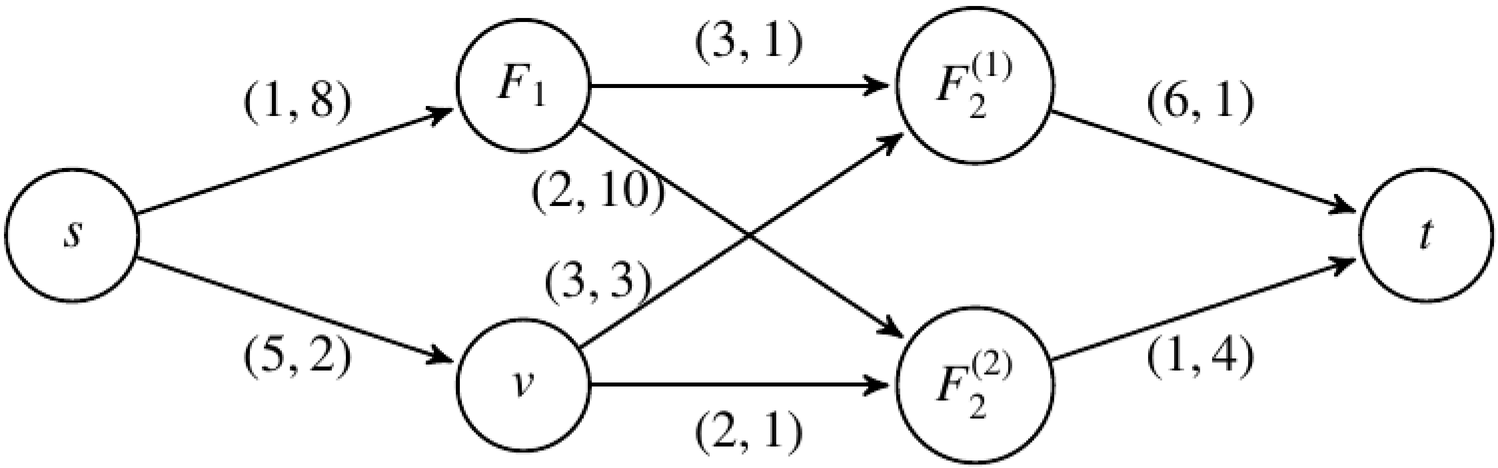
At s, can either go to F1 or v. Update cost and min banwidth accordingly.

$$
\begin{aligned}
t ::= \quad & sw = s; (sw \leftarrow F_1; co \leftarrow co + 1; ca \leftarrow \min\{ca, 8\} \\
& \qquad \& \ sw \leftarrow v; co \leftarrow co + 5; ca \leftarrow \min\{ca, 2\}) \\
& \& \ sw = F_1; \\
& \qquad (sw \leftarrow F_2^{(1)}; co \leftarrow co + 3; ca \leftarrow \min\{ca, 1\} \\
& \qquad \& \ sw \leftarrow F_2^{(2)}; co \leftarrow co + 2; ca \leftarrow \min\{ca, 10\}) \\
& \& \ sw = v; (sw \leftarrow F_2^{(1)}; co \leftarrow co + 3; ca \leftarrow \min\{ca, 3\} \\
& \qquad \& \ sw \leftarrow F_2^{(2)}; co \leftarrow co + 2; ca \leftarrow \min\{ca, 1\}) \\
& \& \ sw = F_2^{(1)}; sw \leftarrow t; co \leftarrow co + 6; ca \leftarrow \min\{ca, 1\} \\
& \& \ sw = F_2^{(2)}; sw \leftarrow t; co \leftarrow co + 1; ca \leftarrow \min\{ca, 4\}
\end{aligned}
$$

Can model the topology as the union of smaller policies that encode the behavior of each link.

Example: Rate Changing Functions

Function F_2 increases the flow rate by an additive constant



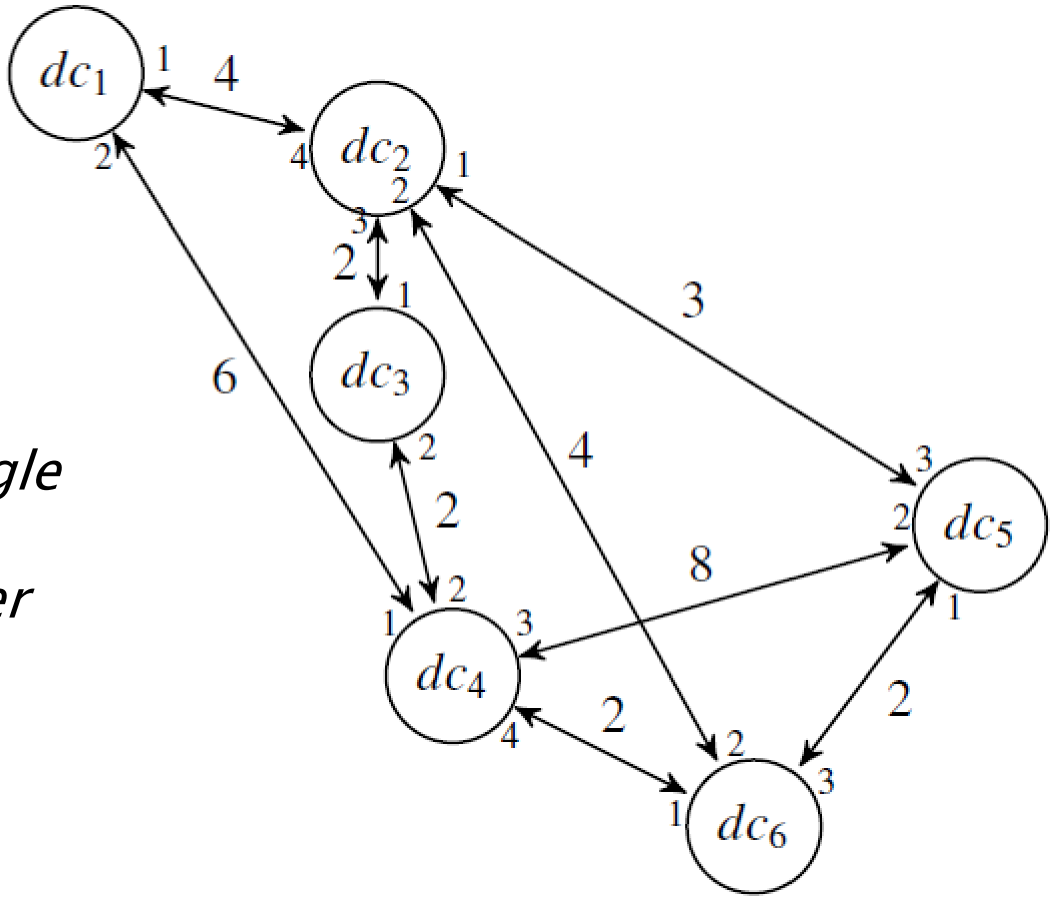
$$p_{F_2} ::= \quad (sw = F_2^{(1)} \,\&\, sw = F_2^{(2)}); ca \leftarrow ca + \gamma$$

Rate changes = capacity changes.

Applications: Cost Reachability

- "Can node B be reached from A at cost at most c?"

Topology: Google B4 Wide-Area Inter-datacenter connect:

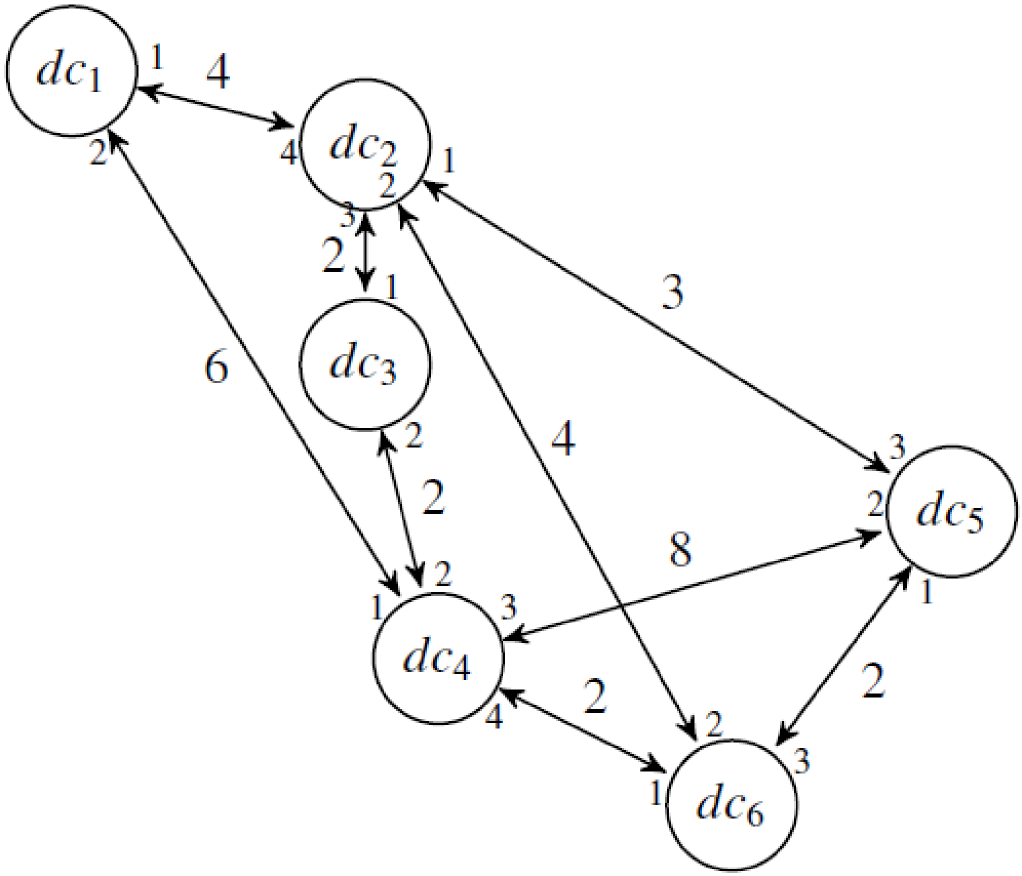


Applications: Capacitated Reachability

- "Can node A communicate with B at rate at least r?"

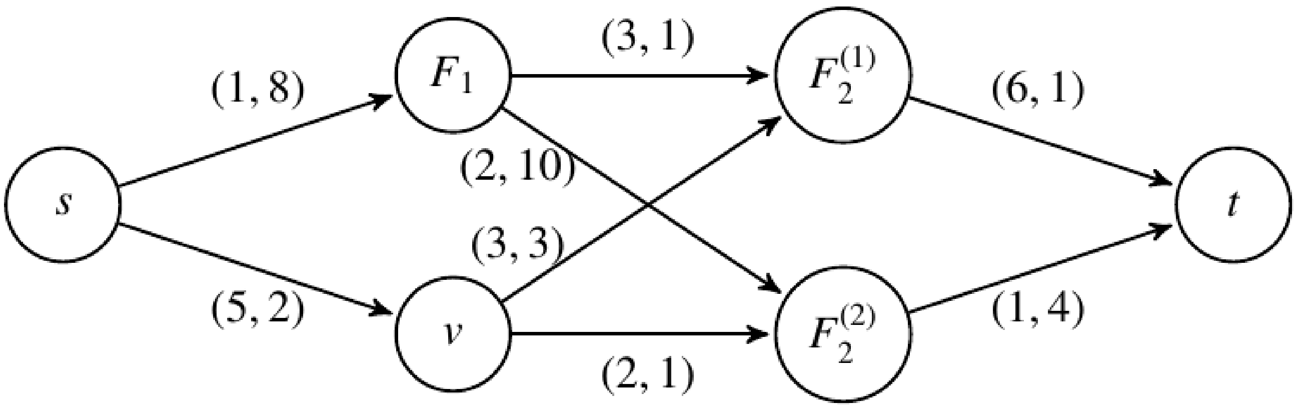
  - Unsplittable

  - Splittable



Applications: Service Chain

- "Can node A reach B at cost/latency at most l and/or at rate/bandwidth at least r, via F1-F2?"



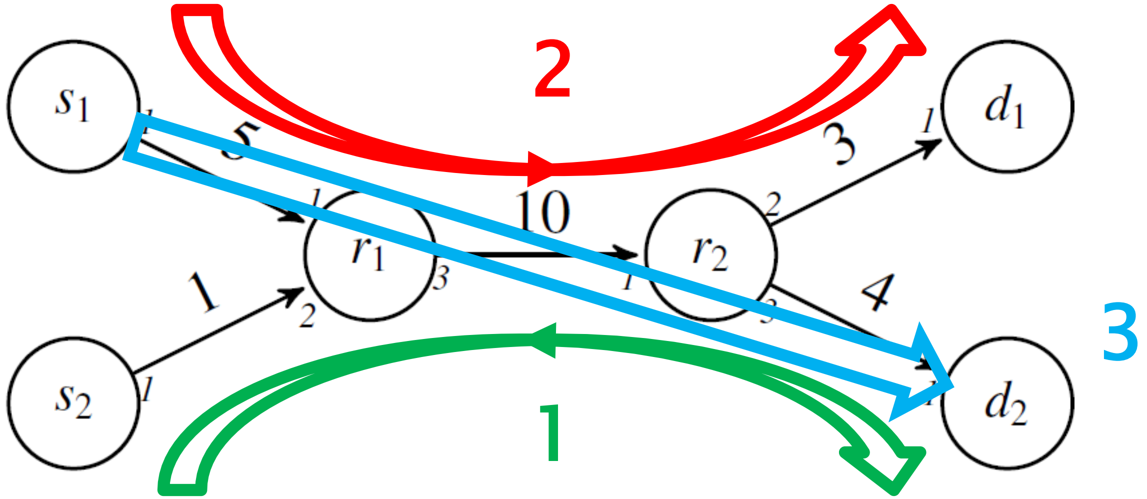
- Check whether the following is equal to "$drop$"

$$src \leftarrow s; dst \leftarrow t; co \leftarrow 0; ca \leftarrow r; sw \leftarrow s;$$
$$pt(pt)^*;$$
$$\boldsymbol{sw = F_1}; p_{F_1}; t(pt)^*; \boldsymbol{sw = F_2}; p_{F_2}; t(pt)^*;$$
$$sw = t; co \leq l; ca \geq r.$$

AALBORG UNIVERSITY

Applications: Fairness

- "Does the current flow allocation satisfy max-min fairness requirements?"



AALBORG UNIVERSITY

(Un)Decidability

- Theorem: (undecidability)

Deciding equivalence of two WNetKAT expressions is equal to deciding the equivalence of the two corresponding weighted WNetKAT automata.

- Theorem:

Deciding whether a WNetKAT expression is equal to "*drop*" is equal to deciding the emptiness of the corresponding weighted automaton.

Questions?

Thank you

for your attention!