wNetKAT: Programming and Verifying Software-Defined Networks

Bingtian Xue
Aalborg University, DENMARK

Joint work with
Kim G. Larsen, Stefan Schmid

NWPT, Nov 1, 2016

Computer Networks

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

- For the future:

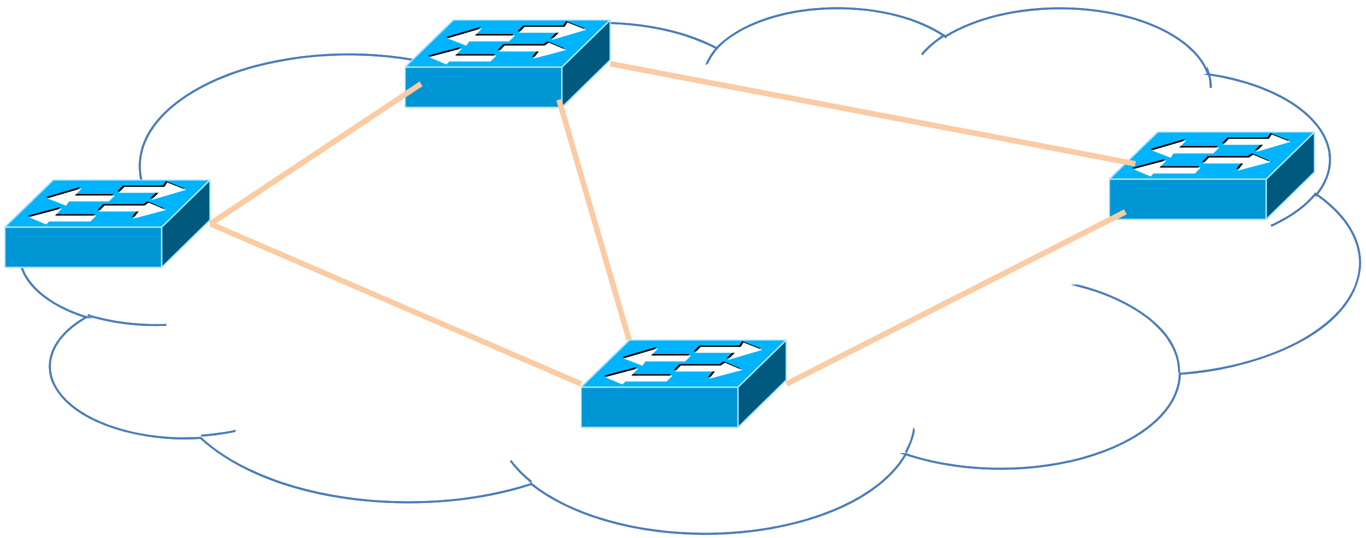
  dependable and flexible enough?

Traditional Networks

Traditional Networks: Data Plane

Data plane:
Packet streaming

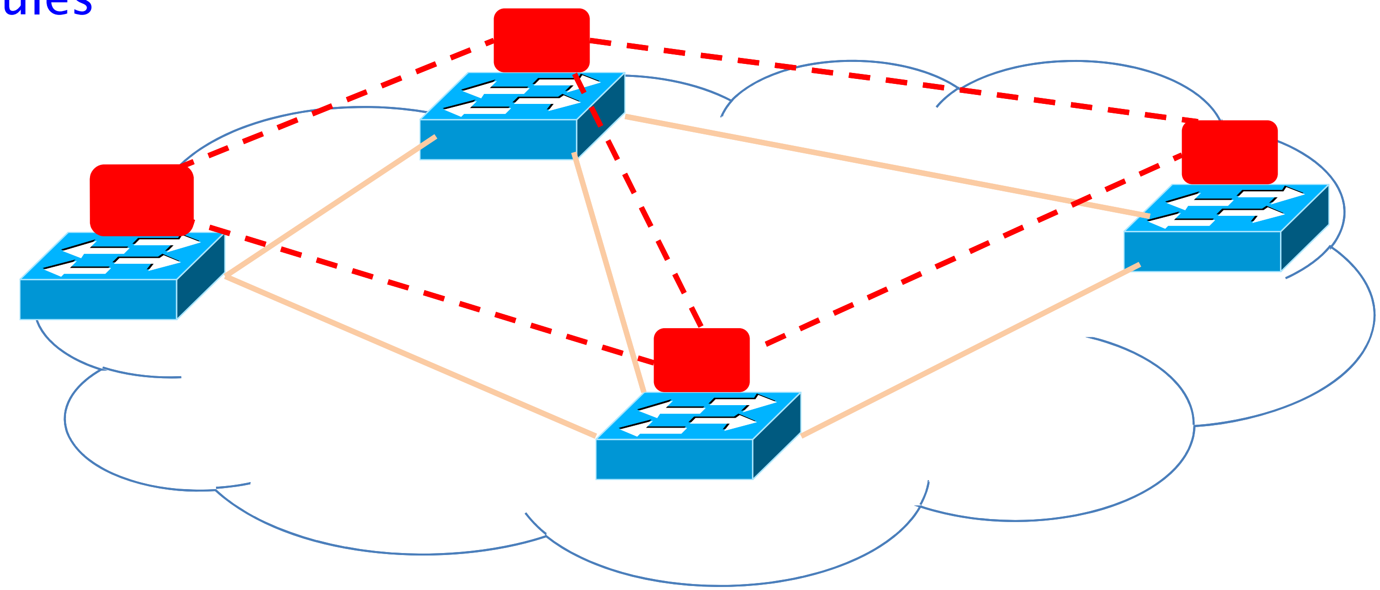
- Forward
- Filter
- Buffer
- Mark
- Rate-limit
- Measure packets



Traditional Networks: (Distributed) Control Plane

Control plane:
Distributed algorithms

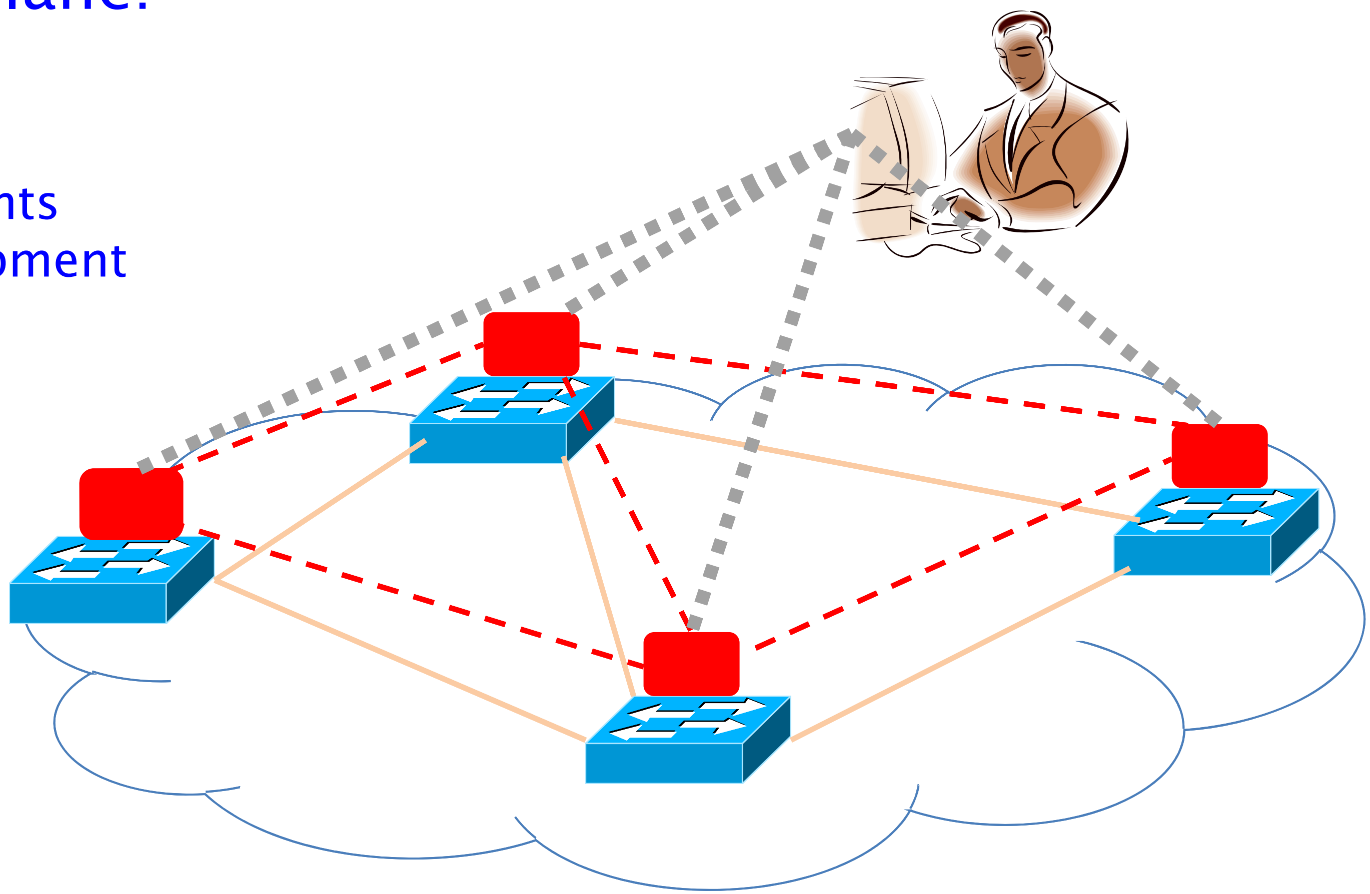
Track topology changes
Compute routes
Install forwarding rules



Traditional Networks: Management Plane

Management plane:
Humans

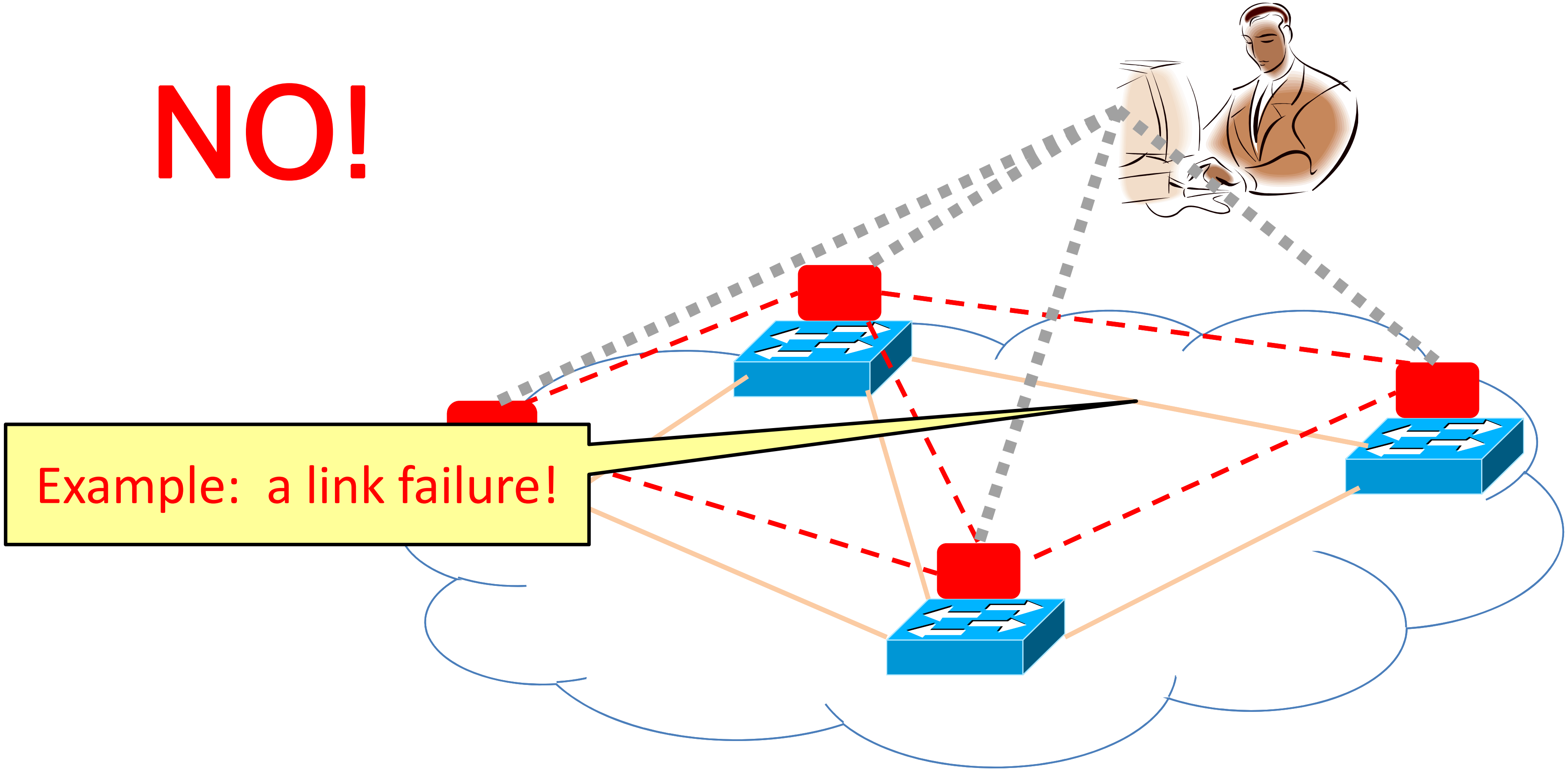
Collect measurements
Configure the equipment



Traditional Networks

- Dependable and flexible enough?

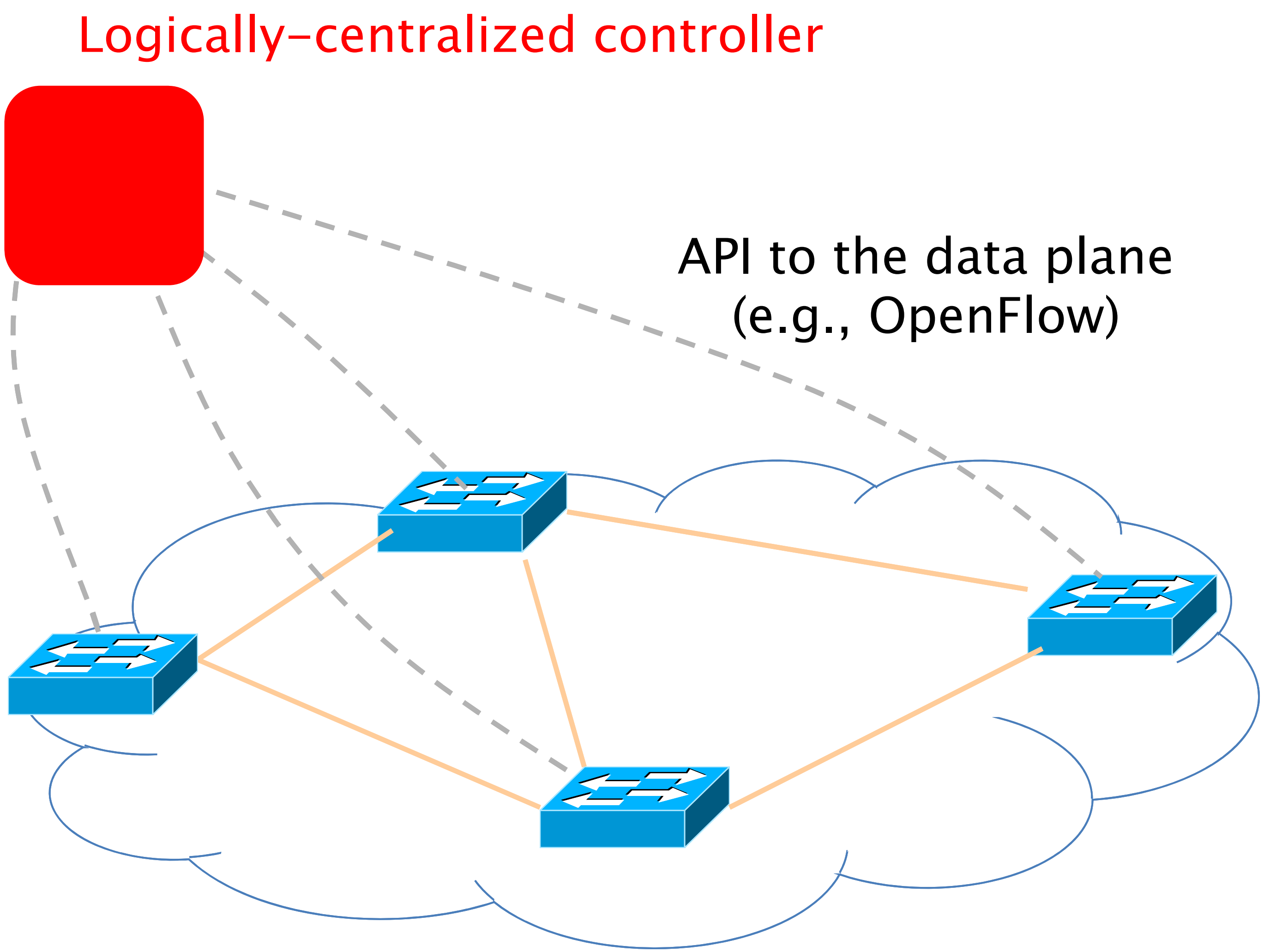
NO!



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

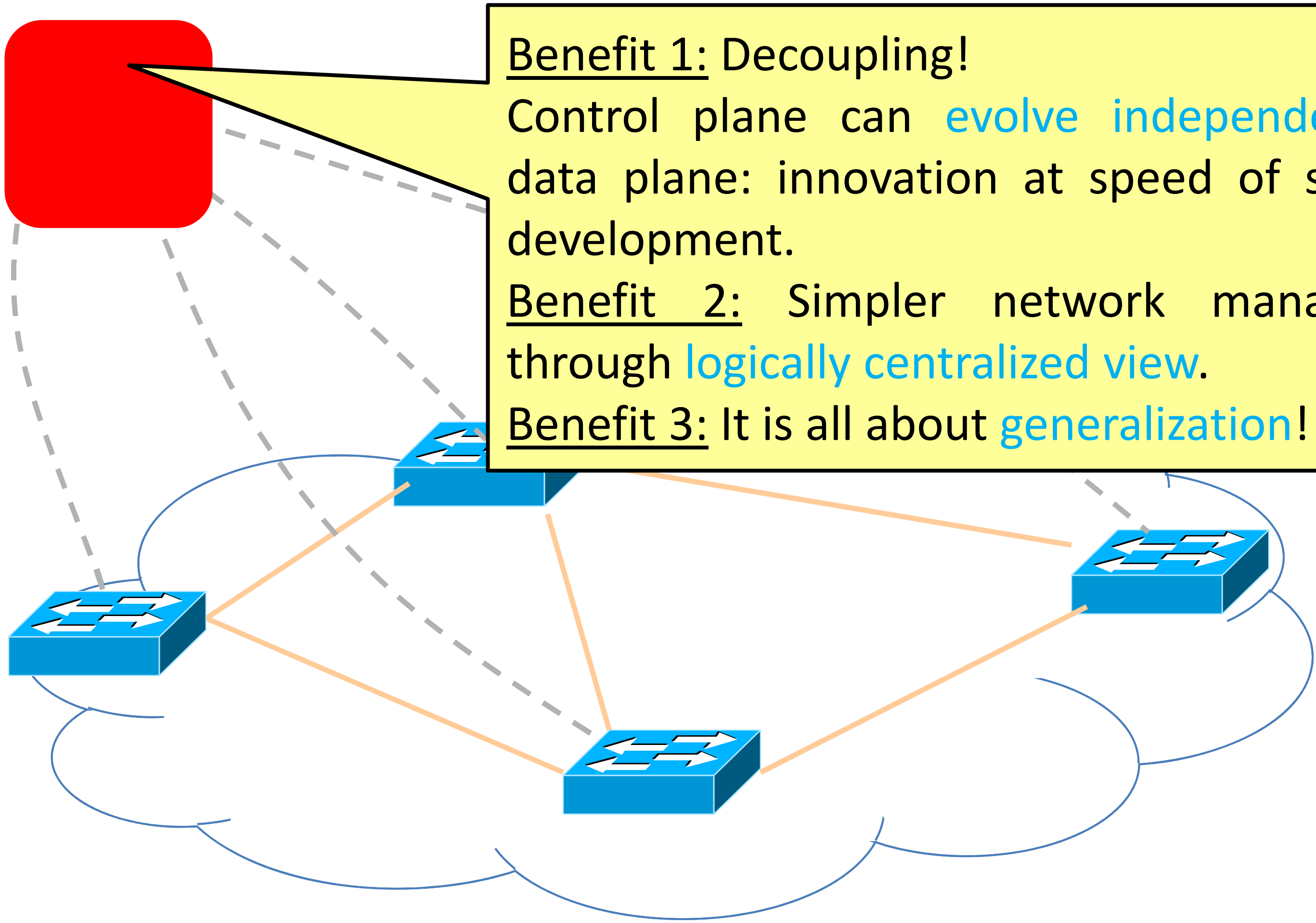
- "A Purpose-Built Global Network: Google's Move to SDN"
  - One of the key reasons for Google's move to SDN: faster and more efficient/fine-grained traffic engineering, improving WAN network utilization.
  - A second reason: more predictable behavior under failures (no unpredictable and slow distributed reconvergence).

Software Defined Networks (SDN)



Software Defined Networks (SDN)

Logically-centralized controller

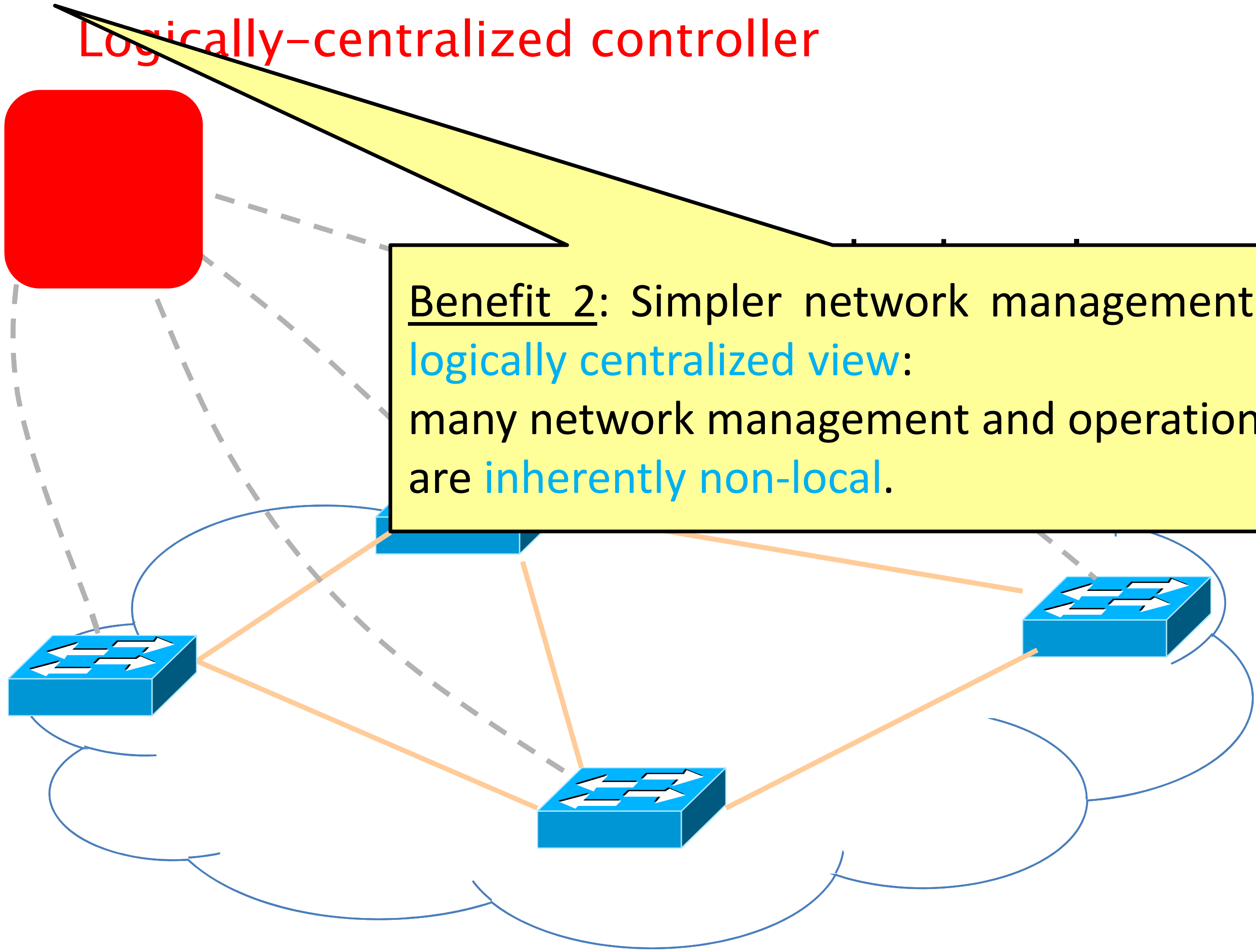


<u>Benefit 1:</u> Decoupling!
Control plane can evolve independently of data plane: innovation at speed of software development.

<u>Benefit 2:</u> Simpler network management through logically centralized view.

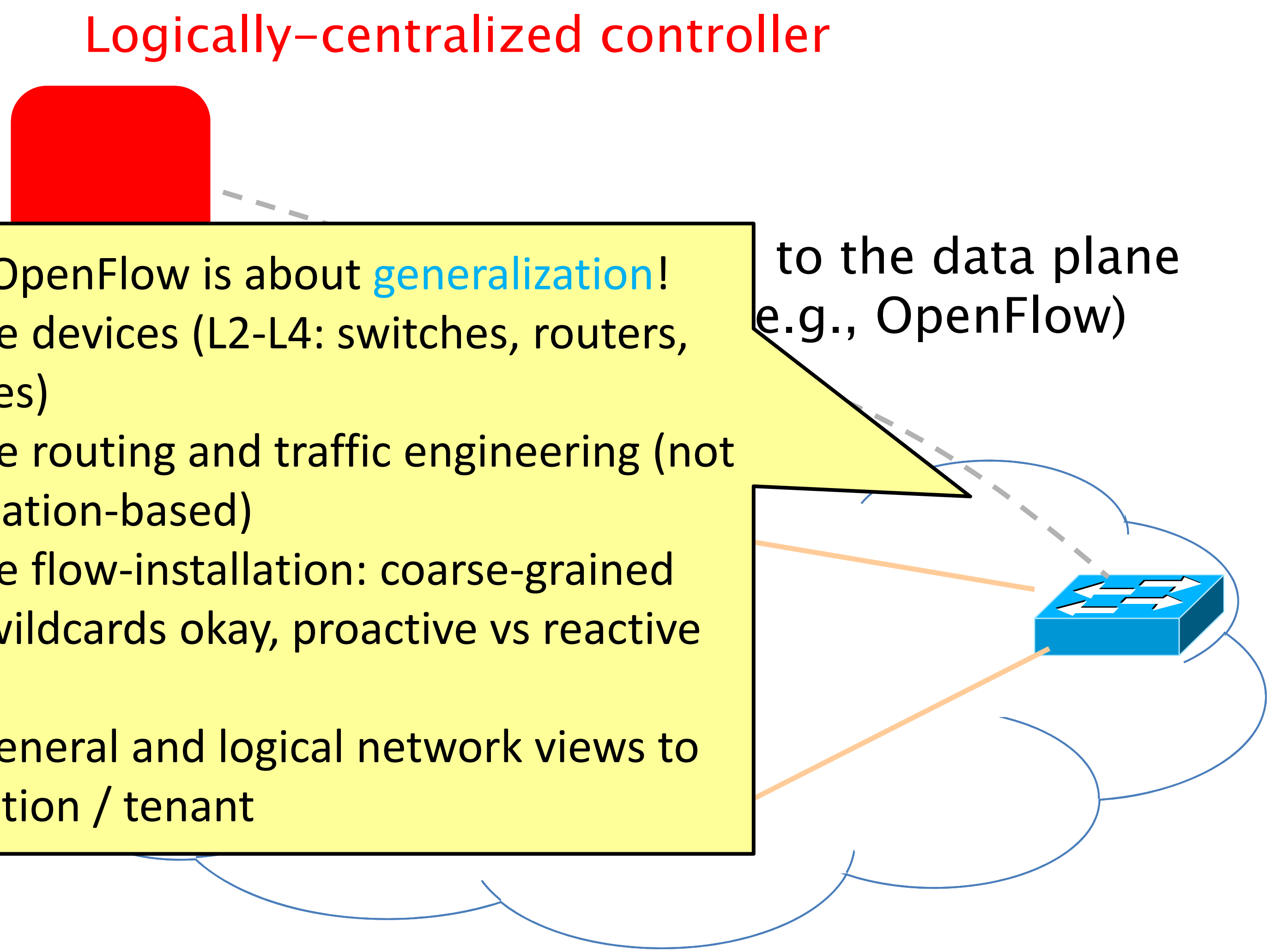
<u>Benefit 3:</u> It is all about generalization!

Software Defined Networks (SDN)



Benefit 2: Simpler network management through logically centralized view:
many network management and operational tasks are inherently non-local.

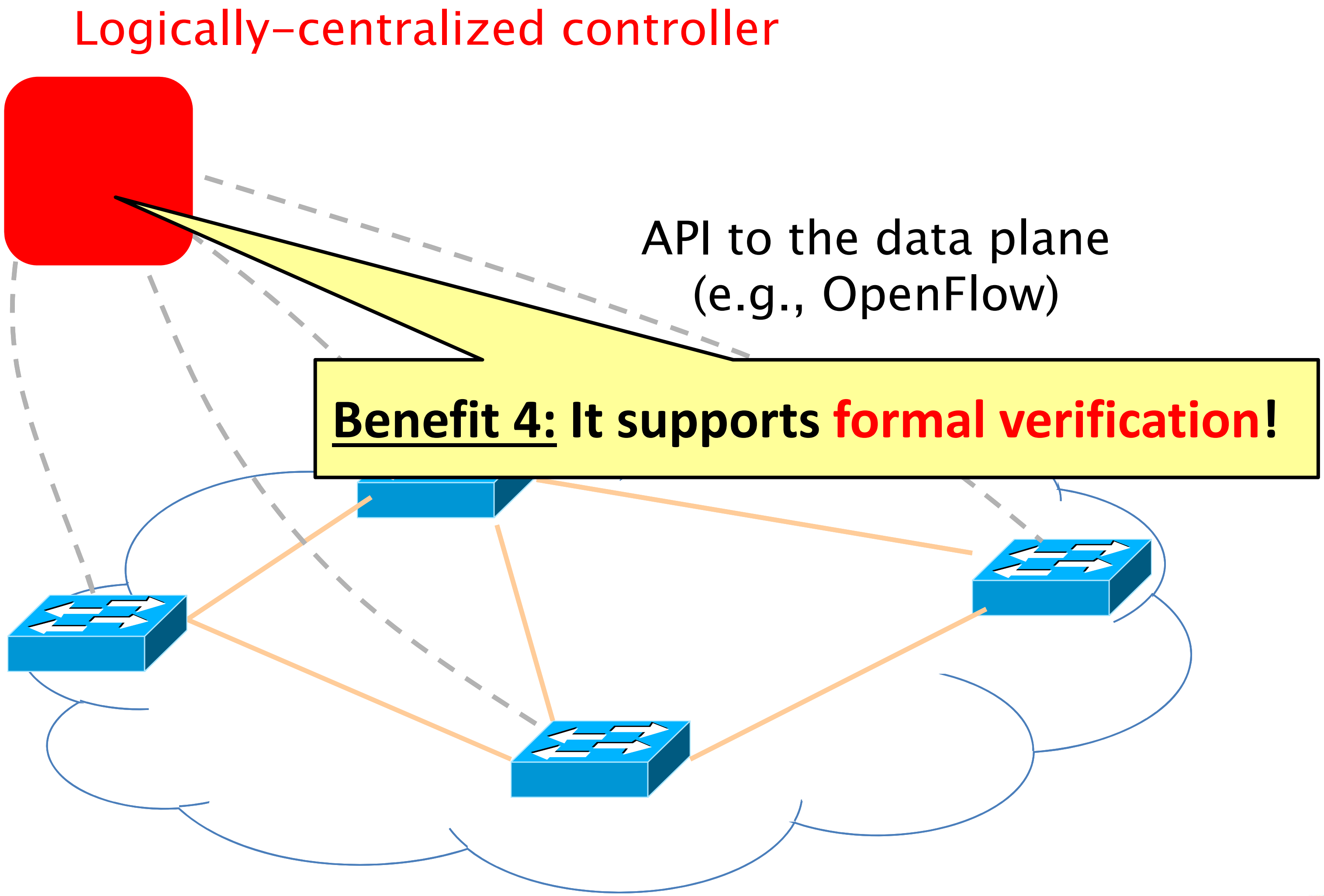
Software Defined Networks (SDN)



<u>Benefit 3:</u> OpenFlow is about generalization!

- Generalize devices (L2-L4: switches, routers, middleboxes)
- Generalize routing and traffic engineering (not only destination-based)
- Generalize flow-installation: coarse-grained rules and wildcards okay, proactive vs reactive installation
- Provide general and logical network views to the application / tenant

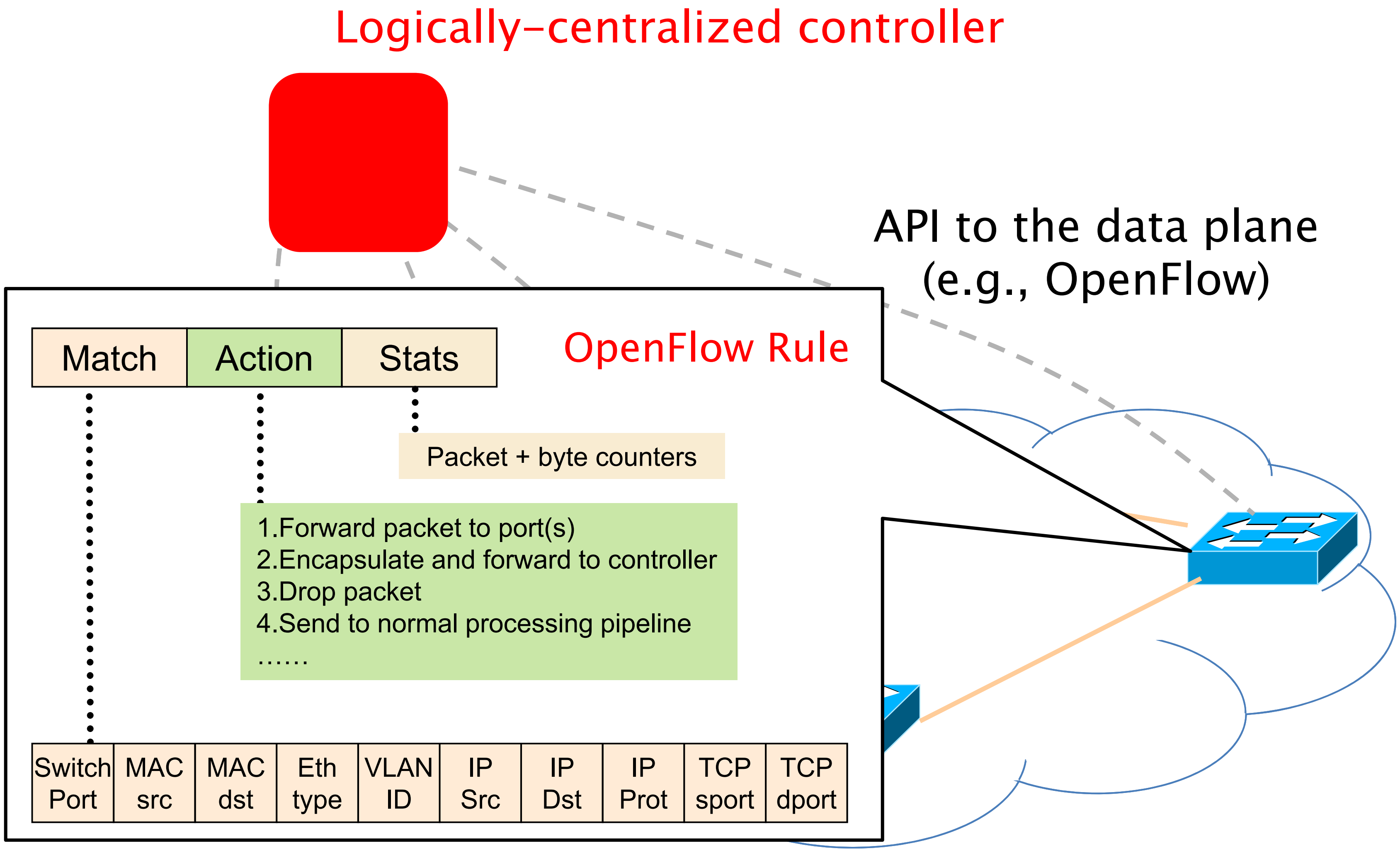
Software Defined Networks (SDN)



Programming SDN

- Programming languages for networks lie at the heart of SDN

Programming SDN



Programming SDN

- Programming languages for networks lie at the heart of SDN
- OpenFlow is very low-level: inconvenient for programmers
- Researchers have started developing more high-level languages
- NetKAT: state-of-the-art framework for programming and reasoning about networks

NetKAT

- Kleene Algebra (KA) $(K, +, \cdot, *, 0, 1)$
  - $+, \cdot$ binary operators
  - $*$ unary operator
  - $0, 1$ constants
- KAT $(K, B, +, \cdot, *, \bar{\ }, 0, 1)$, $B \subseteq K$
  - $(K, +, \cdot, *, 0, 1)$ KA
  - $(B, +, \cdot, \bar{\ }, 0, 1)$ Boolean Algebra
  - $(B, +, \cdot, 0, 1)$ subalgebra of $(K, +, \cdot, 0, 1)$
- NetKAT: KAT with the following atoms
  - $f \leftarrow w$ assignment
  - $f = w$ test
  - ***dup*** duplication

NetKAT

Syntax

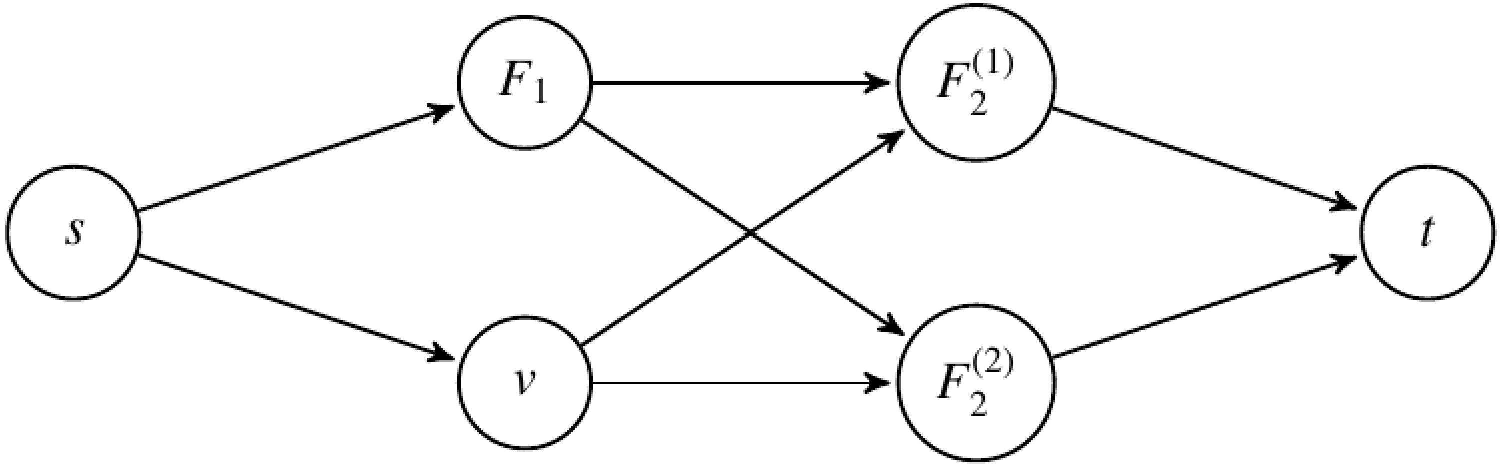
$$
\begin{array}{rrll}
\text{Fields} & f ::= & f_1 \mid \cdots \mid f_k & \\
\text{Packets} & pk ::= & \{f_1 = v_1, \cdots, f_k = v_k\} & \\
\text{Histories} & h ::= & pk::\langle\rangle \mid pk::h & \\
\text{Predicates} & a, b ::= & 1 & \textit{Identity} \\
& \mid & 0 & \textit{Drop} \\
& \mid & f =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

Semantics

$$
\begin{array}{c}
[\![p]\!] \in \mathsf{H} \to \mathcal{P}(\mathsf{H}) \\
[\![1]\!]\ h \triangleq \{h\} \\
[\![0]\!]\ h \triangleq \{\} \\
[\![f = n]\!]\ (pk::h) \triangleq \begin{cases} \{pk::h\} & \text{if } pk.f = n \\ \{\} & \text{otherwise} \end{cases} \\
[\![\neg a]\!]\ h \triangleq \{h\} \setminus ([\![a]\!]\ h) \\
[\![f \leftarrow n]\!]\ (pk::h) \triangleq \{pk[f := n]::h\} \\
[\![p + q]\!]\ h \triangleq [\![p]\!]\ h \cup [\![q]\!]\ h \\
[\![p \cdot q]\!]\ h \triangleq ([\![p]\!] \bullet [\![q]\!])\ h \\
[\![p^*]\!]\ h \triangleq \bigcup_{i \in \mathbb{N}} F^i\ h \\
\text{where } F^0\ h \triangleq \{h\} \text{ and } F^{i+1}\ h \triangleq ([\![p]\!] \bullet F^i)\ h \\
[\![\mathsf{dup}]\!]\ (pk::h) \triangleq \{pk::(pk::h)\}
\end{array}
$$

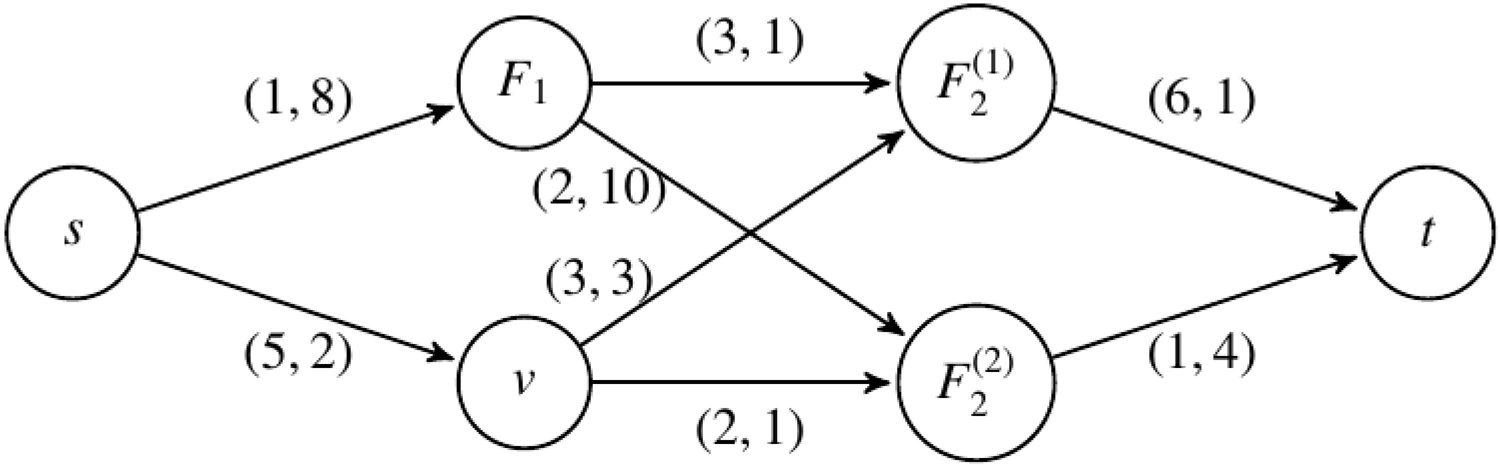
';' instead of '·' in policies!

NetKAT: Example



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

NetKAT: Example



Weighted NetKAT!

wNetKAT

- Add quantitative variables!
- Add switch variables (quantitative, non-quantitative)
- Quantitative Assignment

$$x \leftarrow (\Sigma_{x' \in \mathcal{V}'} x' + \delta)$$

- Quantitative Test

$$x \bowtie (\Sigma_{x' \in \mathcal{V}'} x' + \delta)$$

where: $x \in \mathcal{V}_q, \mathcal{V}' \subseteq \mathcal{V}_q, \bowtie \in \{>, <, \geq, \leq, =\}, \delta \in \mathbb{N}$ (or $\mathbb{Q}$)

x might be from $\mathcal{V}'$

$\mathcal{V}_q$ ($\mathcal{V}_n$) can be either packet or switch!

& instead of NetKAT +

wNetKAT

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

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

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

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

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

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

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

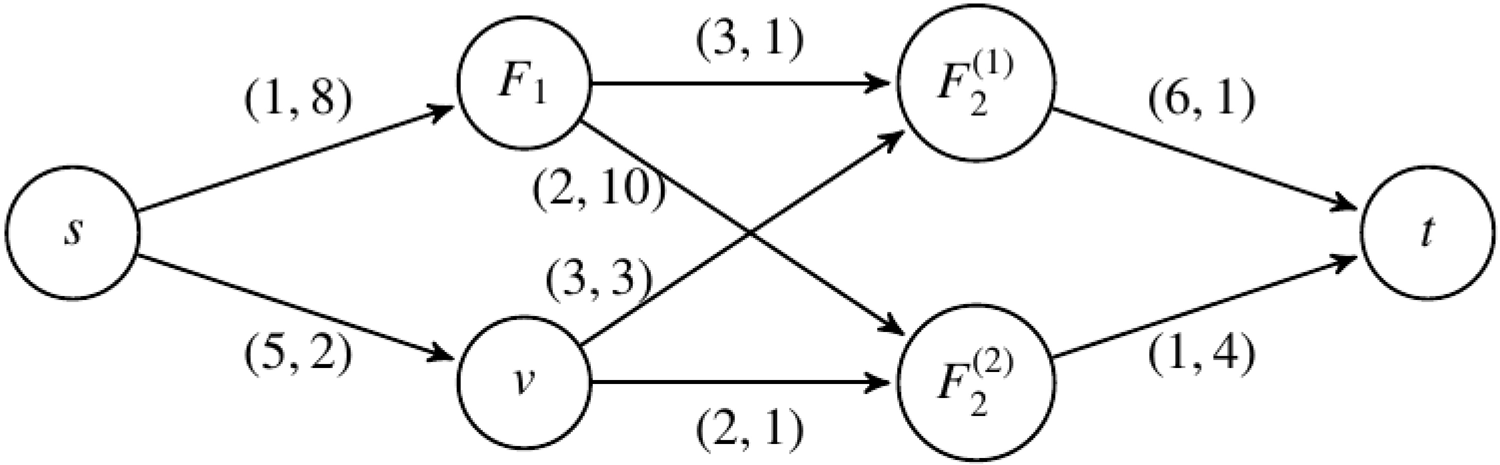
wNetKAT

- Other arithmetic operations

- min or max can be easily defined:

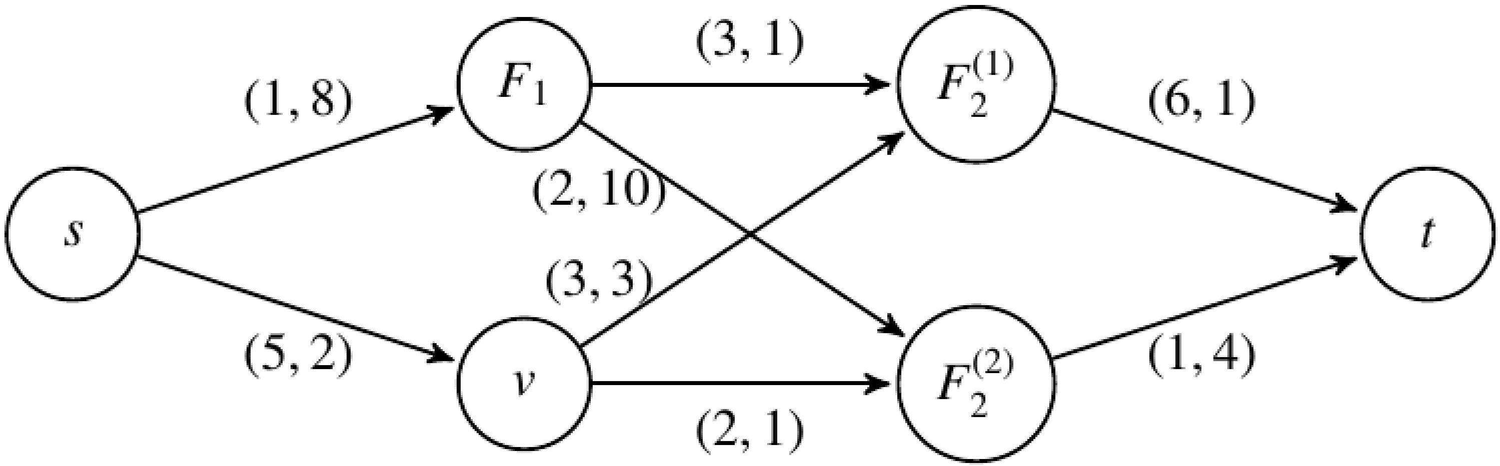
$$x \leftarrow \min\{y, z\} \overset{\text{def}}{=} y \leq z; x \leftarrow y \& y > z; x \leftarrow z.$$

Example



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

Example

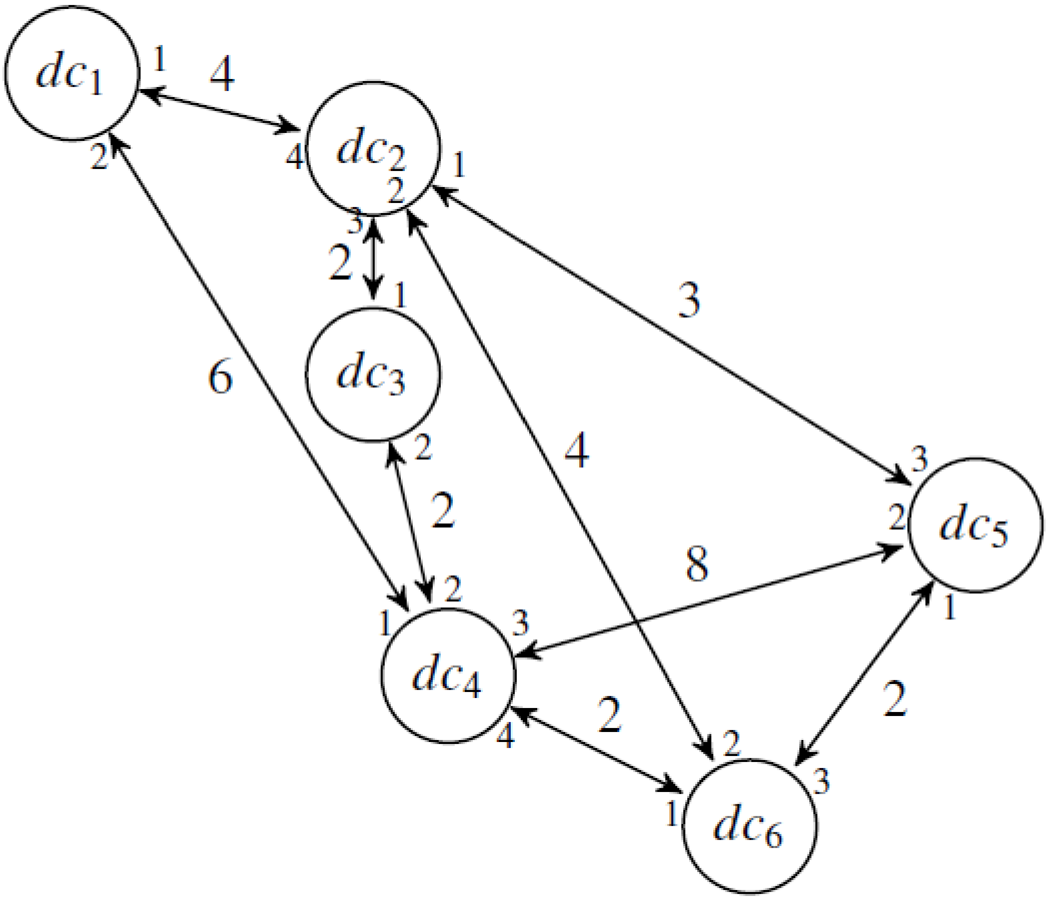


$$p_{F_2} ::= \quad (sw = F_2^{(1)} \ \& \ sw = F_2^{(2)}); ca \leftarrow ca + \gamma$$

Only packet variables in this example

Applications: Cost Reachability

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



Applications: Cost Reachability

- "Can node B be reached from A at cost at most c?"
  - Topology $'\boldsymbol{t}'$

  e.g., $dc_1 \longrightarrow dc_2$

  $sw = dc_1; pt = 1; sw \leftarrow dc_2; pt \leftarrow 4; l \leftarrow l + 4$

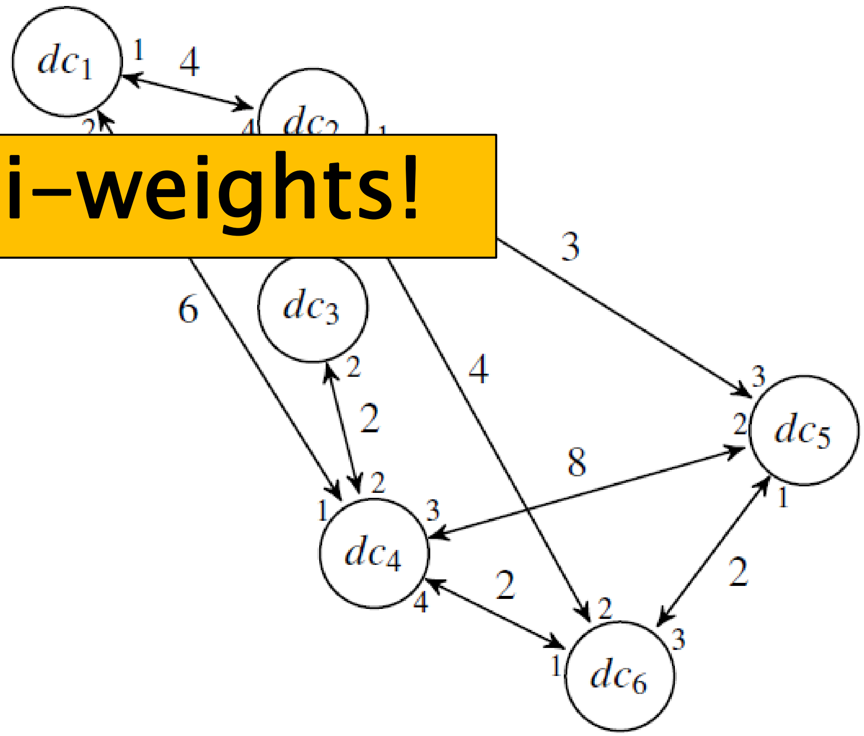
  - Policy $'\boldsymbol{p}'$

  e.g., dc_2

  sr

  $(p$

Also works for multi-weights!



  - Check whether the following

  equal to "$drop$"

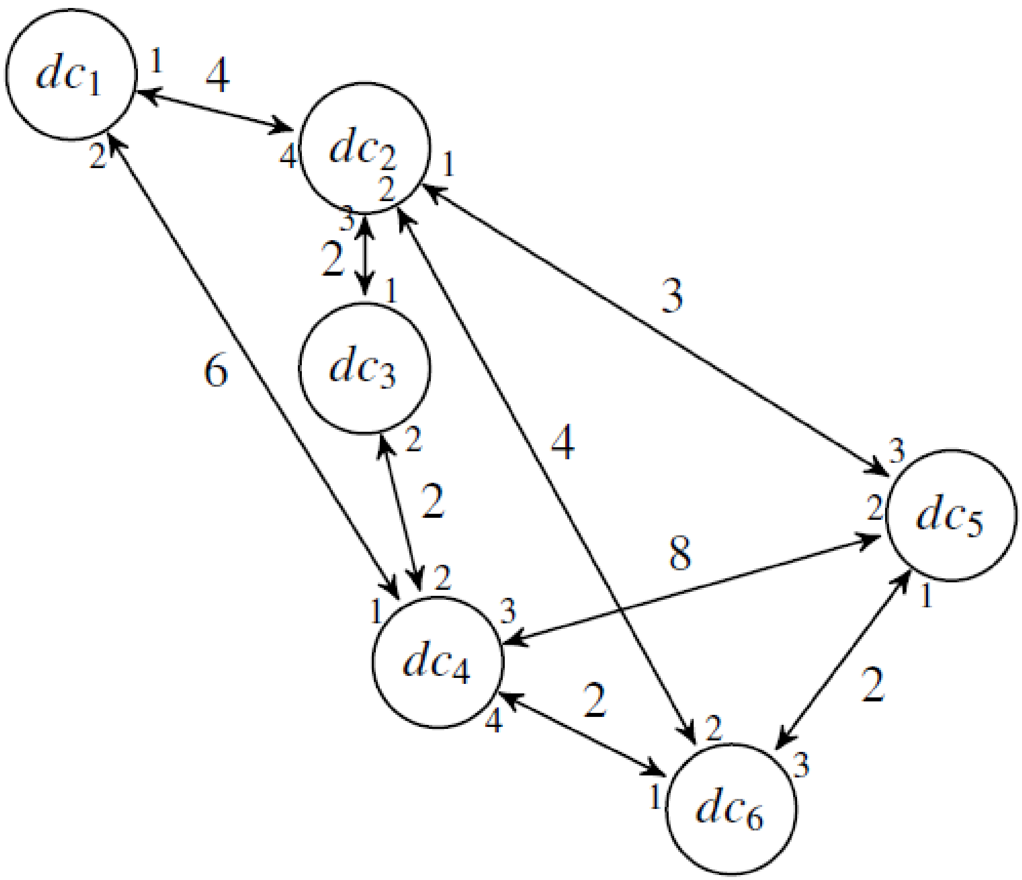
  $scr \leftarrow A; dst \leftarrow B; l \leftarrow 0; sw \leftarrow A;$
  $pt(pt)^*;$
  $sw = B; l \leq c.$

Applications: Capacitated Reachability

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

  - Unsplittable

  - Splittable



Applications: Capacitated Reachability

- "Can node A communicate with B at rate at least r?"
- Unsplittable
  - Topology $'\boldsymbol{t}'$

    e.g., $dc_1 \longrightarrow dc_2$

    $sw = dc_1; pt = 1;$

    $sw \leftarrow dc_2; pt \leftarrow 4;$ $c \leftarrow \min\{c, 4\}$
  - Policy $'\boldsymbol{p}'$

    e.g., dc_2

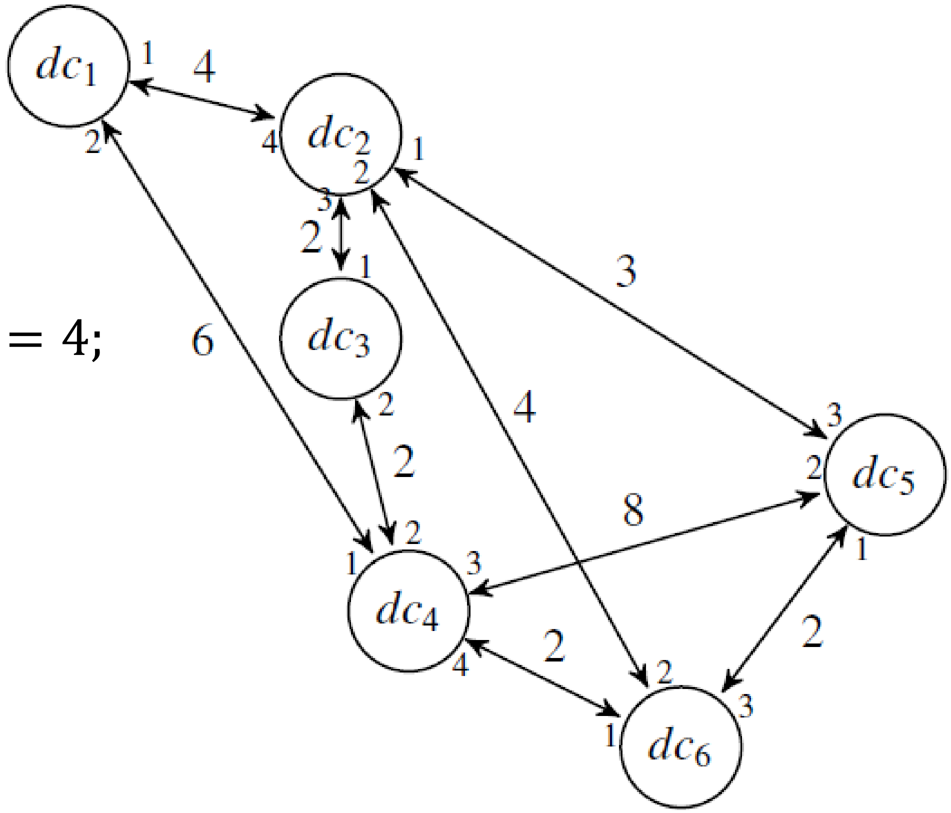
    $src = dc_1; dst = dc_5; sw = dc_2; pt = 4;$

    $(pt \leftarrow 1 \;\&\; pt \leftarrow 3)$
  - Check whether the following equal to "$drop$"

    $scr \leftarrow A; dst \leftarrow B;$ $c \leftarrow r;$ $sw \leftarrow A;$

    $pt(pt)^*;$

    $sw = B;$ $c \geq r$.



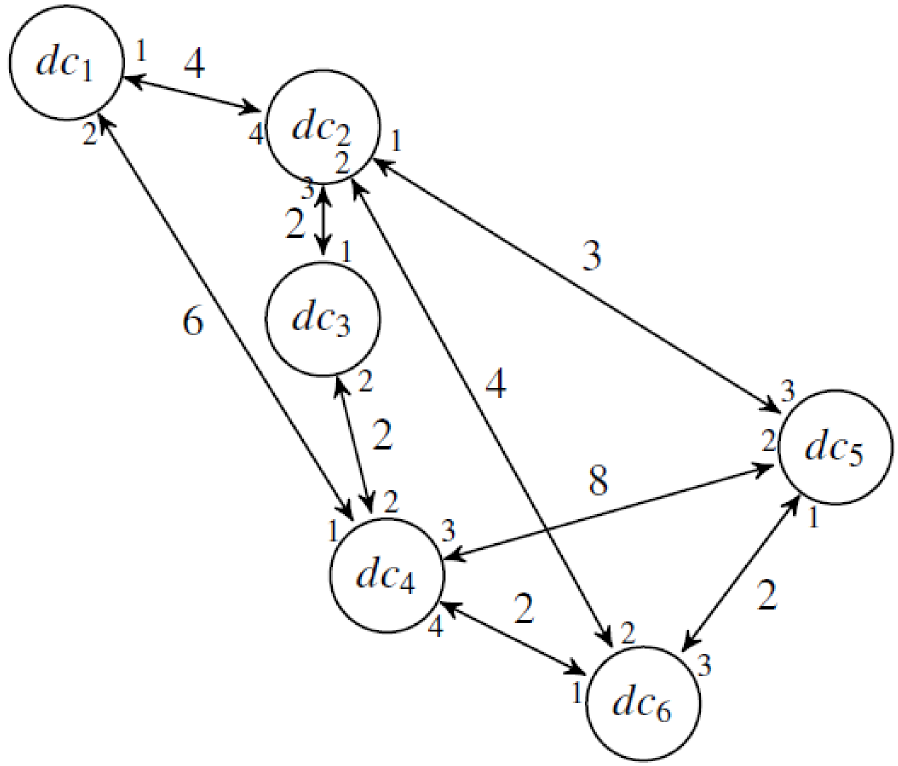
Applications: Capacitated Reachability

- "Can node A communicate with B at rate at least r?"
- Splittable
  - Topology $'\boldsymbol{t}'$

  e.g., $dc_1 \longrightarrow dc_2$

  $sw = dc_1; pt = 1;$

  $sw \leftarrow dc_2; pt \leftarrow 4; c \leftarrow \min\{c, 4\}$

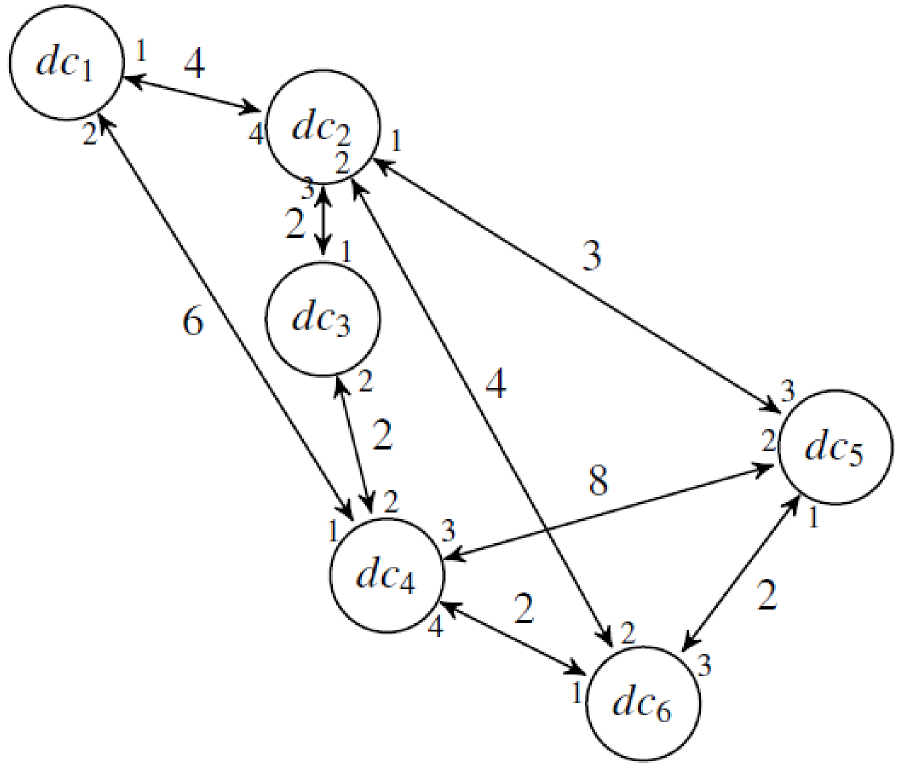


AALBORG UNIVERSITY

Applications: Capacitated Reachability

- "Can node A communicate with B at rate at least r?"
- Splittable
  - Policy $'p'$
  - Split: e.g., dc_2

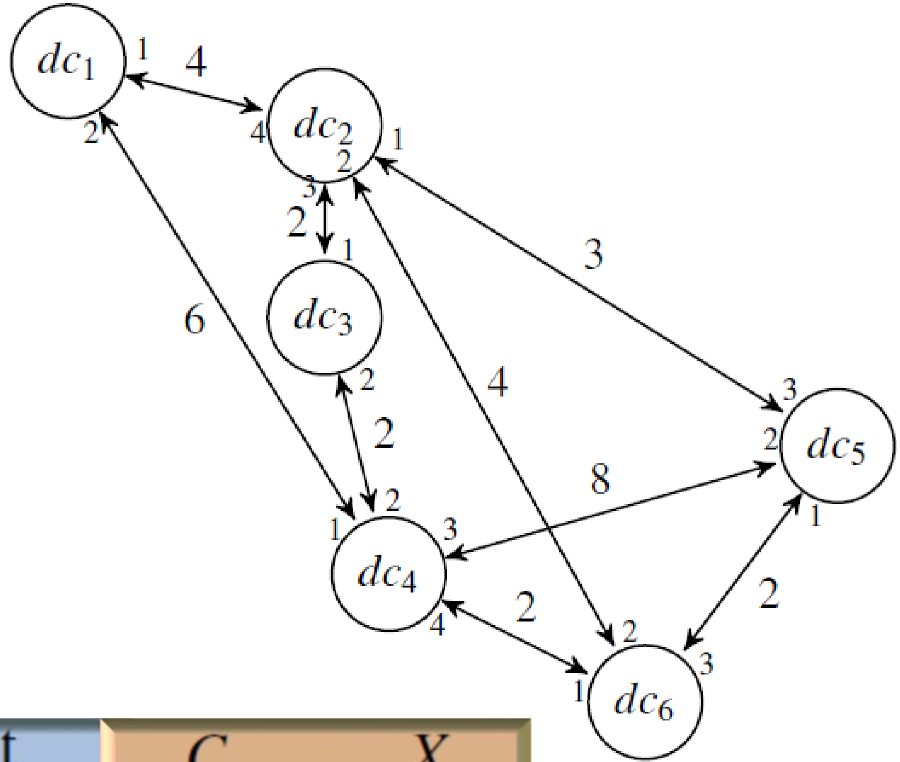
$src = dc_1; dst = dc_5; sw = dc_2;$
$pt = 4; c \leq 5;$
$((pt \leftarrow 1\ ; c \leftarrow \min\{c, 3\})$
$\&\ (pt \leftarrow 3; c \leftarrow \max\{0, c - 3\}))$



Applications: Capacitated Reachability

- "Can node A communicate with B at rate at least r?"
- Splittable
  - Policy $'p'$
  - Split
  - Merge: e.g., dc_4

$sw = dc_4; src = dc_1; dst = dc_5;$
$(pt = 1 \,\&\, pt = 2);$
$C = C + c; X \leftarrow X - 1;$
$(X \neq 0; drop$
$\&\, X = 0; c \leftarrow C; pt \leftarrow 3).$

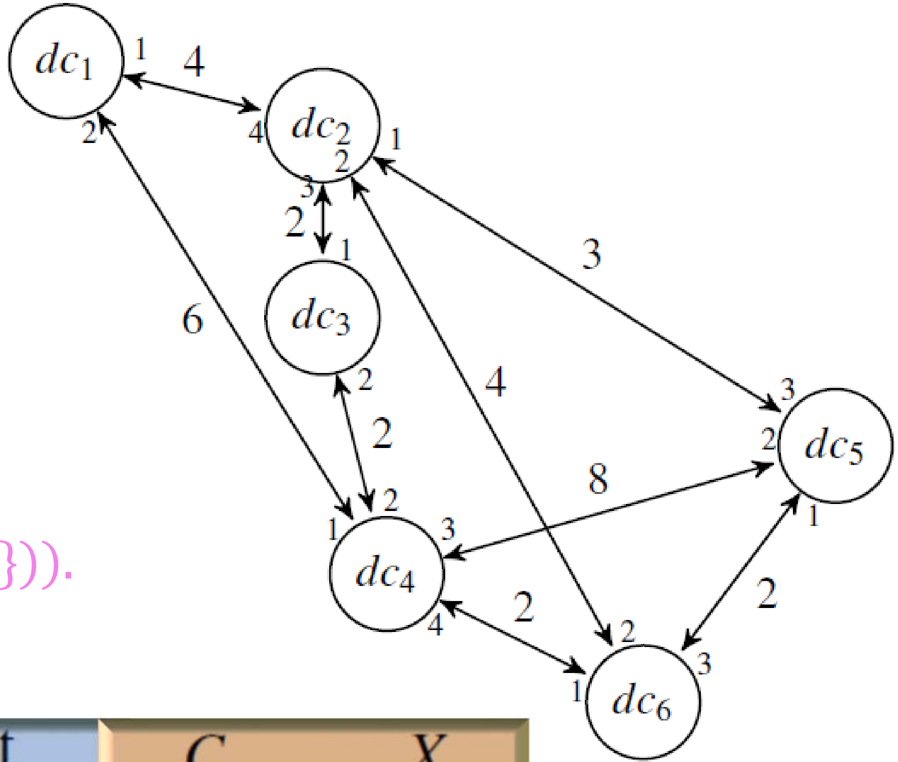


src	dst	in	out	C	X
dc_1	dc_5	1, 2	3	0	2
dc_5	dc_2	3, 4	1, 2	0	2

Applications: Capacitated Reachability

- "Can node A communicate with B at rate at least r?"
- Splittable
  - Policy $'p'$

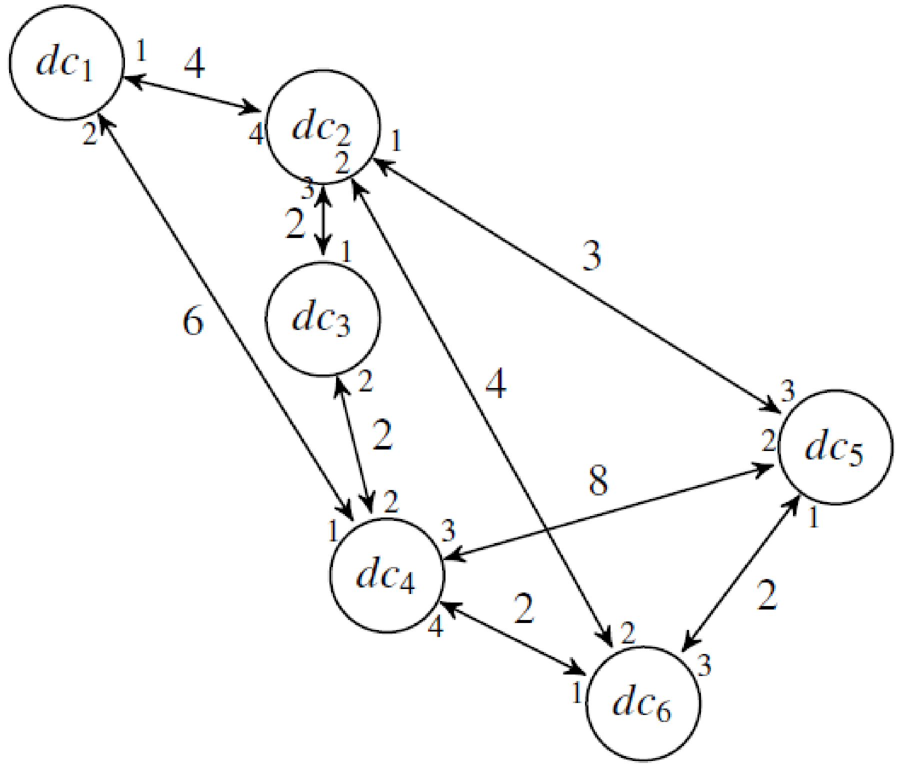
$sw = dc_4; src = dc_5; dst = dc_2;$
$(pt = 3 \,\&\, pt = 4);$
$C = C + c; X \leftarrow X - 1;$
$(X \neq 0; drop$
$\&\, X = 0; c \leftarrow C; c \leq 8$
$((pt \leftarrow 1; c \leftarrow \min\{6, c\})$
$\&\, (\mathrm{pt} \leftarrow 2; c \leftarrow \max\{0, c - 6\})).$



src	dst	in	out	C	X
dc_1	dc_5	1, 2	3	0	2
dc_5	dc_2	3, 4	1, 2	0	2

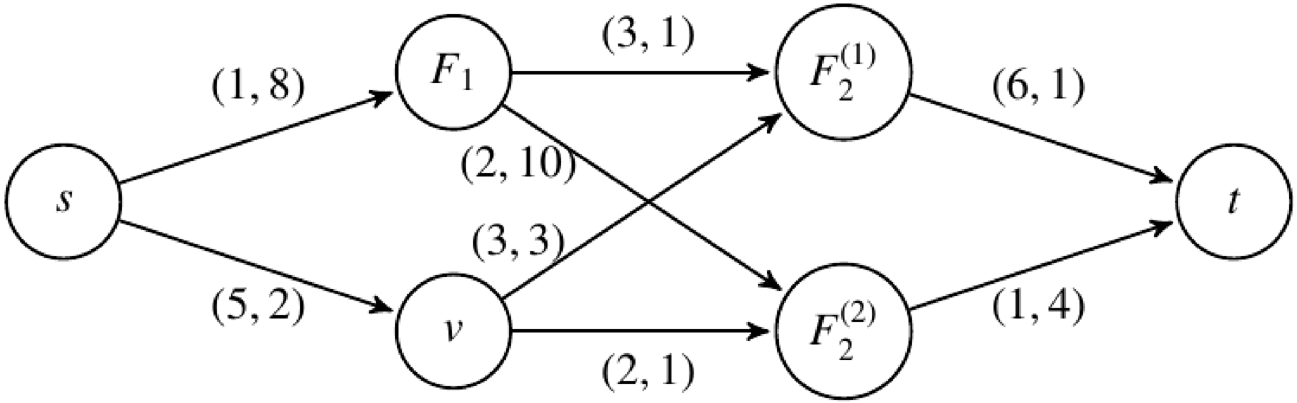
Applications: Capacitated Reachability

- “Can node A communicate with B at rate at least r?”
- Splittable
  - Check whether the following equal to "$drop$"
    $scr \leftarrow A; dst \leftarrow B; c \leftarrow r; sw \leftarrow A;$
    $pt(pt)^*;$
    $sw = B; c \geq r.$



Applications: Service Chain

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

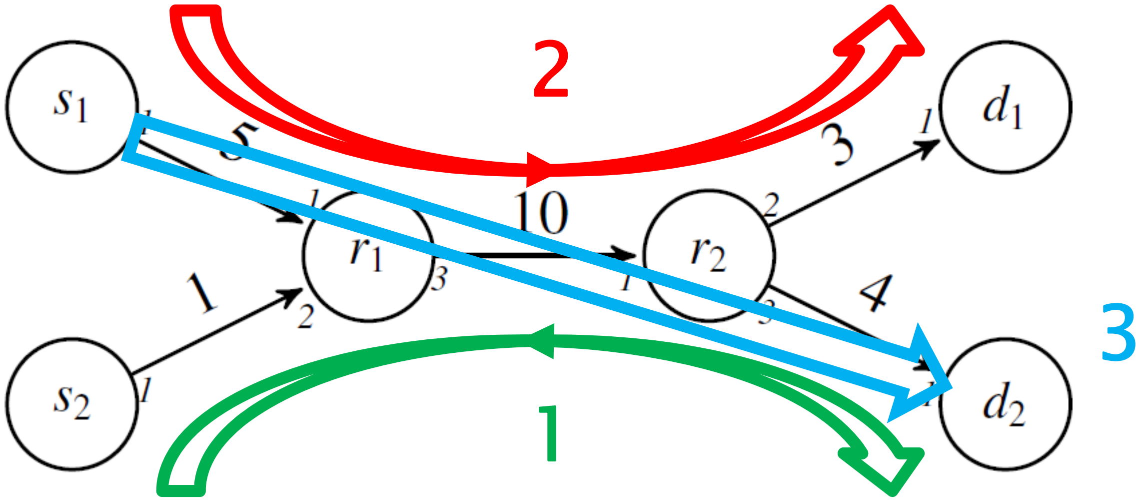


- Check whether the following equal to "$drop$"

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

Applications: Fairness

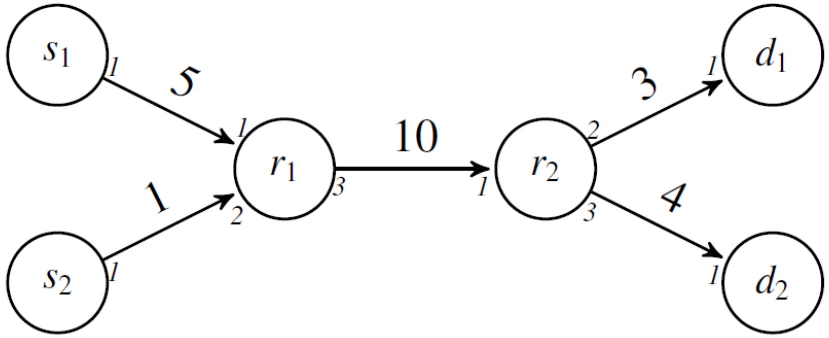
- "Does the current flow allocation satisfy max–min fairness requirements?"



AALBORG UNIVERSITY

Applications: Fairness

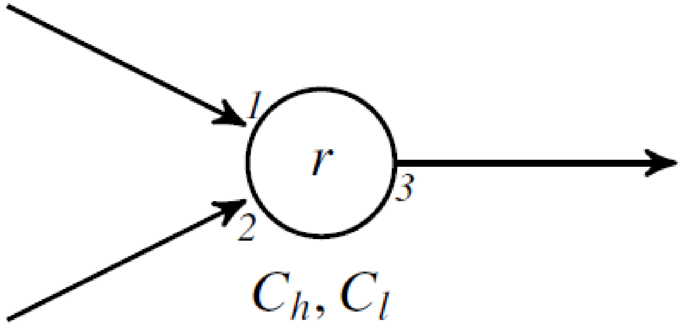
- "Does the current flow allocation satisfy max-min fairness requirements?"



$$
\begin{aligned}
f_3 &= sw \leftarrow s_2; scr \leftarrow s_2; dst \leftarrow d_2; a \leftarrow 10;\\
&\quad x_3 \leftarrow 1; x_1 \leftarrow 0; x_2 \leftarrow 0; tp(tp)^*; sw = d_2; x_3 = a\\
f_1 &= sw \leftarrow s_1; scr \leftarrow s_1; dst \leftarrow d_1; a \leftarrow 10;\\
&\quad x_1 \leftarrow 2; x_3 \leftarrow 1; x_2 \leftarrow 0; tp(tp)^*; sw = d_1; x_1 = a\\
f_2 &= sw \leftarrow s_1; scr \leftarrow s_1; dst \leftarrow d_2; a \leftarrow 10;\\
&\quad x_2 \leftarrow 3; x_1 \leftarrow 2; x_3 \leftarrow 1; tp(tp)^*; sw = d_1; x_2 = a
\end{aligned}
$$

Applications: Quality of Service

- E.g., prioritize a certain flow (e.g., a VoIP call) over another (e.g., a Dropbox synchronization).



$$
\begin{aligned}
sw = r; (pt = 1 \;\&\; pt = 2);\\
&(x = high; C_h < 8; pt \leftarrow 3; C_h \leftarrow C_h + 1;\\
&\& \; x = low; C_l < 2; pt \leftarrow 3; C_l \leftarrow C_l + 1);\\
&C_h = 8; C_l = 2; C_h \leftarrow 0; C_l \leftarrow 0
\end{aligned}
$$

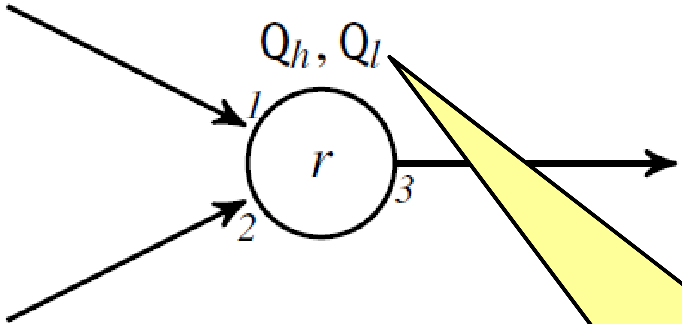
Further Extension: Queues

$$[\![\mathsf{EQ}\ \mathtt{Q}]\!](\rho,\ pk :: h) \quad = \quad \begin{cases} [\![1^*]\!](\rho,\ pk :: h) & \text{if } \mathtt{Q} \neq FULL, \\ & \text{then } EQ(\mathtt{Q}) \\ \emptyset & \text{otherwise} \end{cases}$$

$$[\![\mathsf{DQ}\ \mathtt{Q}]\!](\rho,\ pk :: h) \quad = \quad \begin{cases} \{\rho,\ pk :: h\} & \text{if } HEAD(\mathtt{Q}) = pk :: h \\ & \text{then } DQ(\mathtt{Q}) \\ [\![1^*]\!](\rho,\ pk :: h) & \text{otherwise} \end{cases}$$

Applications: Quality of Service

- E.g., prioritize a certain flow (e.g., a VoIP call) over another (e.g., a Dropbox synchronization).



Coming soon!

$$
\begin{aligned}
&sw = r; x = low; \mathsf{EQ}\ \mathsf{Q}_l\ \&\ sw = \\
&\quad sw = r; x = high; \\
&\qquad (x_h < 8; \mathsf{DQ}\ \mathsf{Q}_h; pt \leftarrow 3; x_h \leftarrow x_h + 1 \\
&\qquad \&\ x_h = 8; Q_l = \emptyset; \mathsf{DQ}\ \mathsf{Q}_h; pt \leftarrow 3 \\
&\qquad \&\ x_h = 8; Q_l \neq \emptyset; skip) \\
&\qquad x_h = 8; x_l = 2; x_h \leftarrow 0; x_l \leftarrow 0
\end{aligned}
$$

(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!