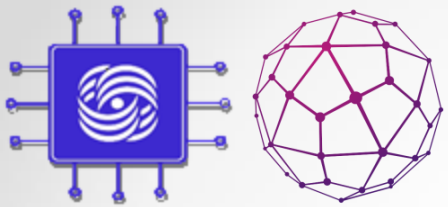


Формальные методы в компьютерных сетях

Доп. главы Компьютерных сетей и телекоммуникации

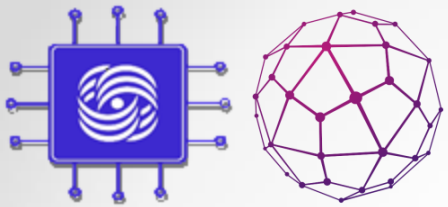
к.ф.-м.н. Чемерицкий Е.В.

Литература:

Qadir, J.; Hasan, O.,

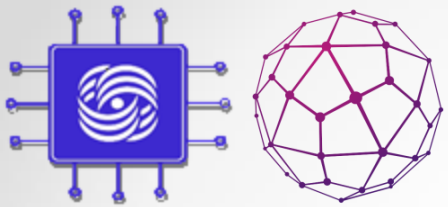
Applying Formal Methods to Networking: Theory, Techniques, and Applications,

Communications Surveys & Tutorials, IEEE , vol.17, no.1, pp.256-291, 2015



Формальные методы

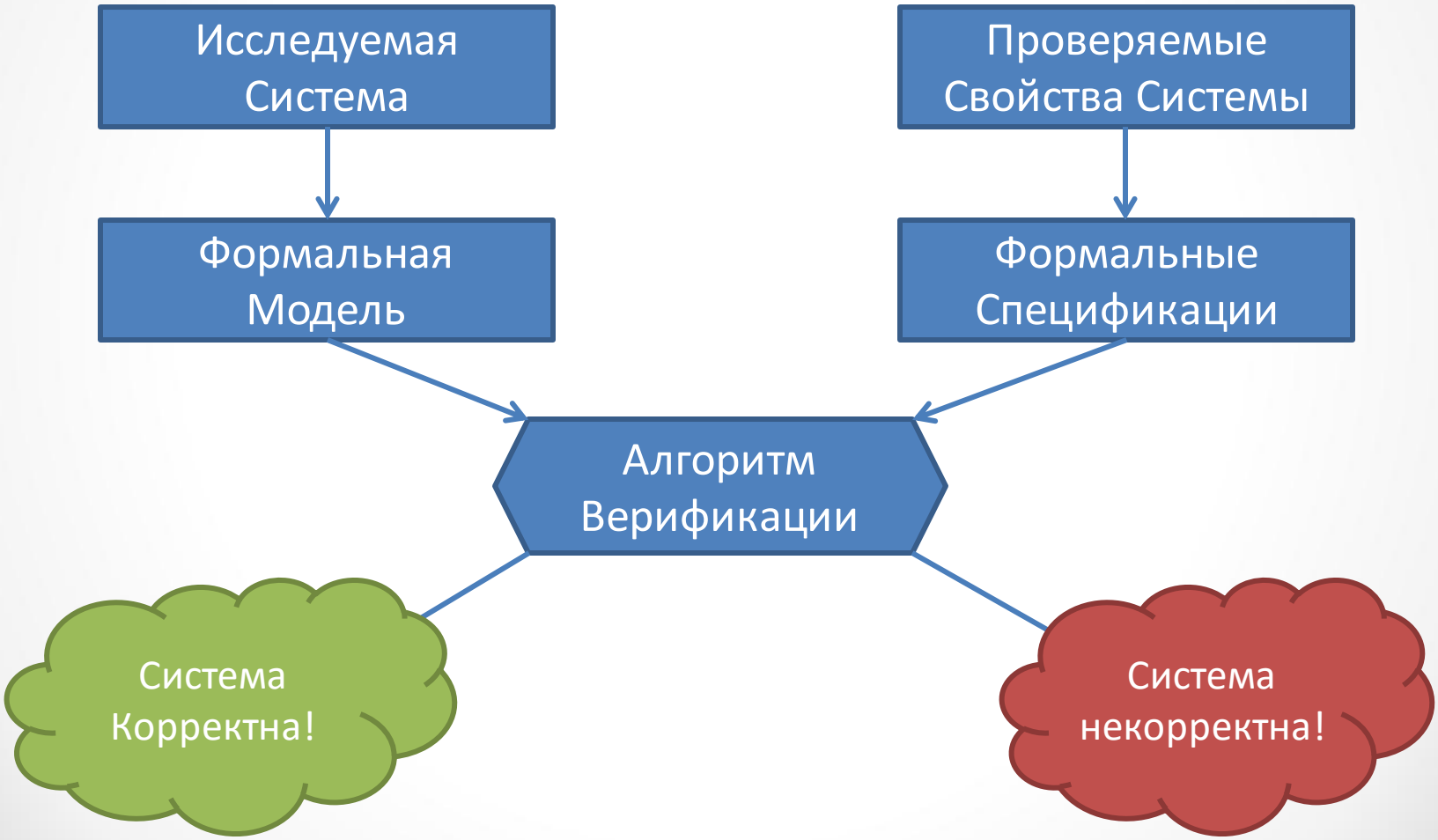
- ***Формальные методы*** – такие методы спецификации, проектирования и анализа свойств систем, которые имеют строгое математическое обоснование
- ***Верификация*** – проверка соответствия свойств системы заданной спецификации
- ***Синтез*** – построения системы, свойства которой удовлетворяют заданной спецификации

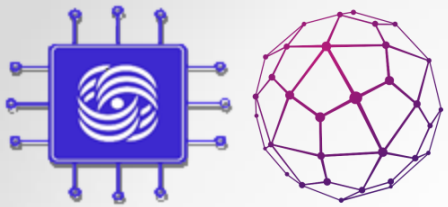


Составные части метода формальной верификации

- ***Модель системы (язык описания модели)*** – математическое представление анализируемой системы
- ***Язык спецификации*** – способ формального описания свойств системы
- ***Метод верификации*** – алгоритм для проверки соответствия модели системы заданной спецификации

Составные части метода формальной верификации





Алгоритм Петерсона (1981)

Исходные данные

Глобальные переменные:

want1 = false, want2 = false, turn = 1

Процесс 1:

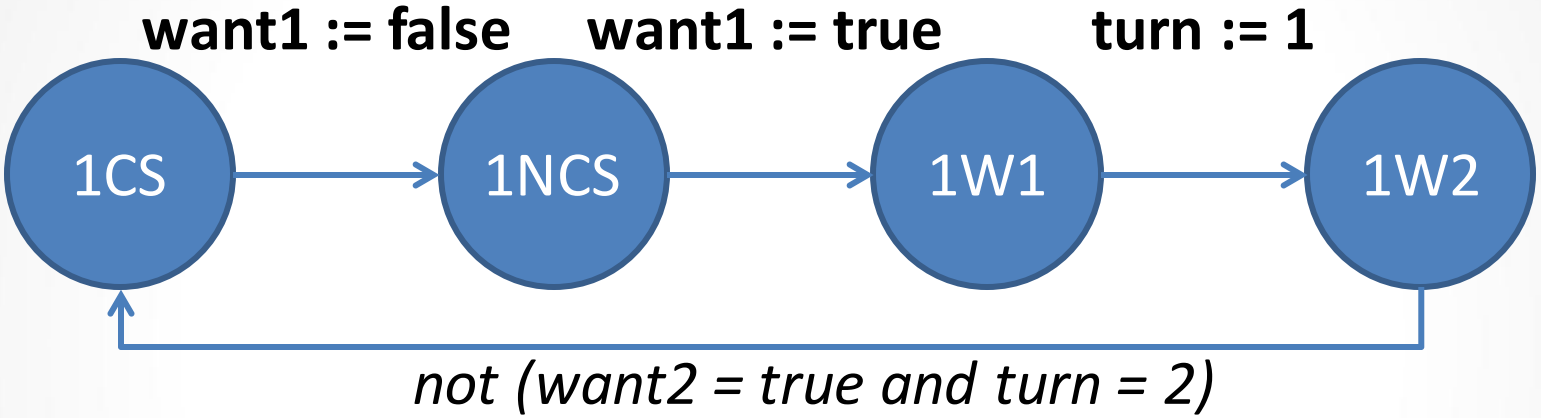
```
while (true) {
   <noncritical section>
   want1 := true;
   turn := 1;
   while want2 and
      turn = 2 do { };
   <critical section>
   want1 := false;
}
```

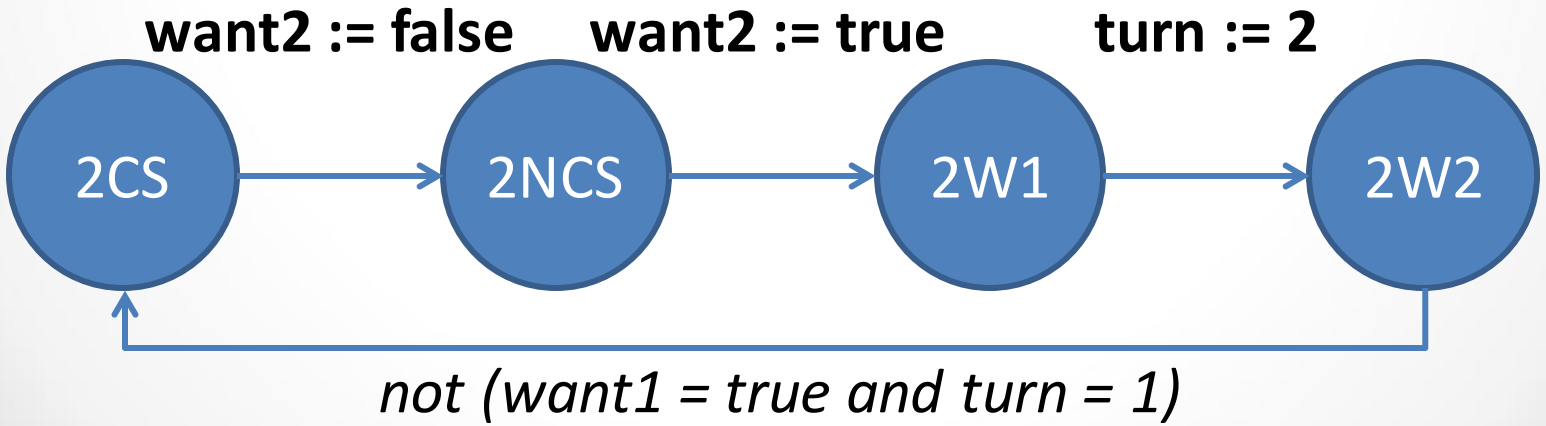
Процесс 2:

```
while (true) {
   <noncritical section>
   want2 := true;
   turn := 2;
   while want1 and
      turn = 1 do { };
   <critical section>
   want2 := false;
}
```

Алгоритм Петерсона (1981)

Формальная модель





Алгоритм Петерсона (1981)

Спецификация свойств

- ***Свойства безопасности (safety)***

  Reachability analysis

  Никогда не произойдёт ситуации, при которой оба процесса одновременно окажутся внутри критической секции

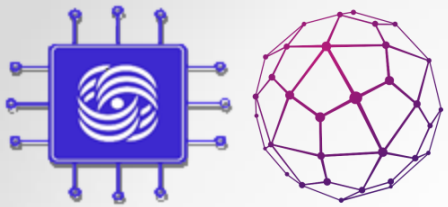
$$\mathbf{G}\neg(1CS \wedge 2CS)$$

- ***Свойства живучести (liveness)***

  Loop detection

  Процесс пожелавший попасть в критическую секцию рано или поздно туда попадёт

$$\mathbf{G}(1NSC \rightarrow \mathbf{F}1CS) \wedge \mathbf{G}(2NSC \rightarrow \mathbf{F}2CS)$$



Алгоритм Петерсона (1981)

Проверка свойств

- Полный перебор диаграммы состояний системы из двух процессов

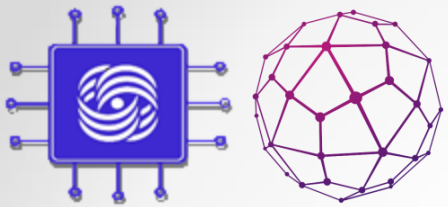
  Сколько состояний будет в системе?

Формальная верификация vs. тестирование

Эффектиность методов	Формальная верификация	Тестирование
Поиск ошибок	средняя	высокая
Доказательство корректности	высокая	низкая
Сложность использования	высокая	низкая

Формальная верификация vs. тестирование

Типы ошибок	Частая	Редкая
Безобидная	тестирование	---
Критическая	тестирование формальная верификация	**формальная верификация**



Типы исследуемых систем

- ***Трансформирующие системы***

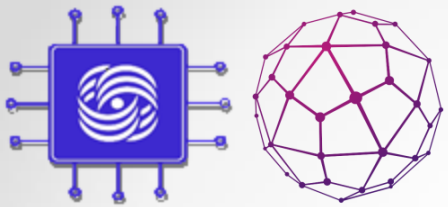
  Преобразовывают входные данные в выходные

  Можно представить таблицей или формулой

- ***Реагирующие системы***

  Поведение зависит от истории воздействий

  Можно представить конечным автоматом



Основные свойства математических моделей

- ***Абстракность***

  Модель должна быть упрощением системы

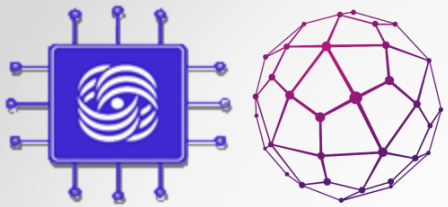
- ***Адекватность***

  Модель должна отражать свойства системы

- ***Компактность***

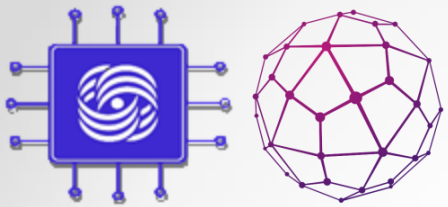
  Размер и структурная сложность модели влияет на сложность решения задачи верификации

  Символьные методы *– модель не перечисляет явным образом все состояния системы*



Некоторые примеры формальных моделей

- Таблицы
- Графовые представления
- Булевы формулы
- Двоичные решающие диаграммы
- Размеченные системы переходов
- Временные автоматы
- Сети Петри



Основные свойства языков спецификации

- ***Выразительная мощность***

  Язык должен охватывать исследуемые свойства

- ***Сложность решения задач***

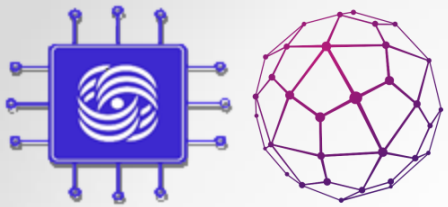
  Проверка эквивалентности формул

  Проверка непротиворечивости формул

  Формальная верификация

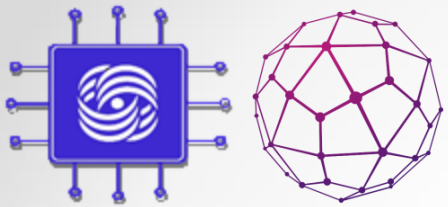
- ***Совместимость с моделью***

  Сложность верификации напрямую зависит от согласованности модели с языком спецификации



Некоторые примеры языков спецификации

- Пропозициональная логика
- Логика предикатов первого порядка
- Логика высших порядков
- Логика Хоара
- Модальная логика
- Темпоральная логика
- Mu-исчисления



Методы формальной верификации

- Неавтоматические

  Доказательство вручную

- Полуавтоматические

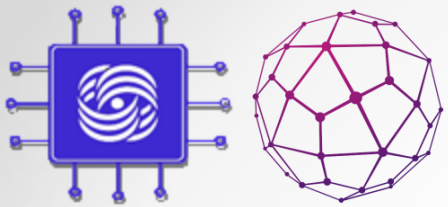
  Theorem proving

  полуразрешимость

- Автоматические

  Model checking

  комбинаторный взрыв



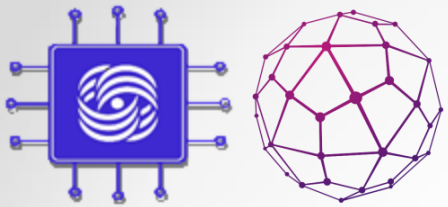
Исторический путь развития компьютерных сетей

“Интернет стал жертвой своего успеха”

- Ценность сети Интернет слишком высока
- Эмпирический закон Меткалфа

Значение для формальных методов:

- Разработка методом проб и ошибок
- Инженерная практика ценится выше теоретических изысканий



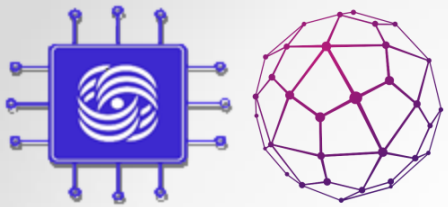
Отцы основатели о формальных методах

- Formal methods have not yielded results commensurate with the effort to use them. They are overblown, verbose, hard to use, hard to understand.

  Vint Cerf

- We reject: kings, presidents and voting.
  We believe in: rough consensus and running code.

  David Clark



Принципы построения компьютерных сетей

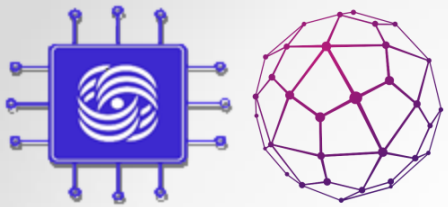
David Clark

The Design Philosophy of the DARPA Internet Protocols

SIGCOMM '88. — ACM, 1988. — Pp. 106–114.

- Отказоустойчивость
- Разнообразие сервисов передачи данных
- Поддержка широкого диапазона сетей
- Распределённое управление ресурсами
- Рентабельность
- Расширяемость
- Учёт использованных сетевых ресурсов

Прогнозирование поведения сети

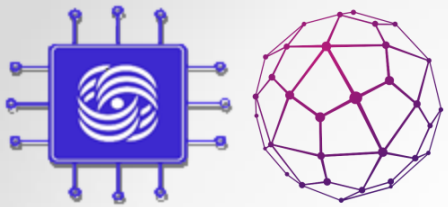


Верификация сетевых протоколов

Исследуемые свойства:

- ***Deadlocks*** - ожидание условий, которые никогда не будут выполнены
- ***Livelocks*** – выполнение инструкций протокола не приближает его к цели
- ***Improper termination*** – протокол завершается не достигнув цели

Число состояний потенциально бесконечно – метод их полного перебора неприменим!



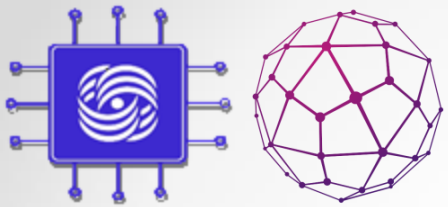
Reachability Analysis

Исследуемые свойства:

- ***Black Holes*** – тихое сбрасывание пакетов
- ***Fowrarding Loops*** – зацикливание пакетов
- Достигнет ли пакет заданной точки?
- Ограничения на длину маршрутов
- Изоляция потоков

Нужен выразительный язык спецификации!

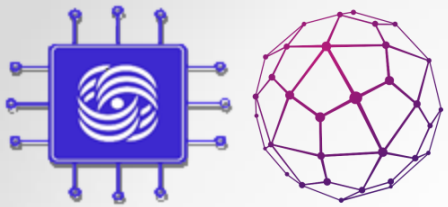
Как восстановить поведение сети?



Network Security

Верификация и синтез конфигураций сетевых экранов:

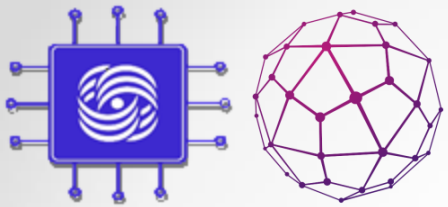
- Проверка эквивалентности
- Проверка избыточности
- Синтез конфигурации состоящей из минимального числа правил



Формальные методы и ПКС

- Новые протоколы управления сетевым оборудованием дали доступ к актуальным правилам обработки пакетов
- Централизация сети позволила собрать информацию о конфигурации сети в единой точке и отслеживать её изменение
- Появилась возможность адаптации наработок из области разработки ПО

Новые сети – новые задачи



Программирование ПКС

Andreas Voellmy, Paul Hudak

Nettle: taking the sting out of programming network routers

Andreas Voellmy, Hyojoon Kim, Nick Feamster

Procera: a language for high-level reactive network control

Joshua Reich , Christopher Monsanto , Nate Foster , Jennifer Rexford , David Walker

Modular SDN Programming with Pyretic

Programming SDN w/ OpenFlow

- The Good
  - Network-wide visibility
  - Direct control over the switches
  - Simple data-plane abstraction

- The Bad
  - Low-level programming interface
  - Functionality tied to hardware
  - Explicit resource control

- The Ugly
  - Non-modular, non-compositional
  - Challenging distributed programming

Программирование ПКС

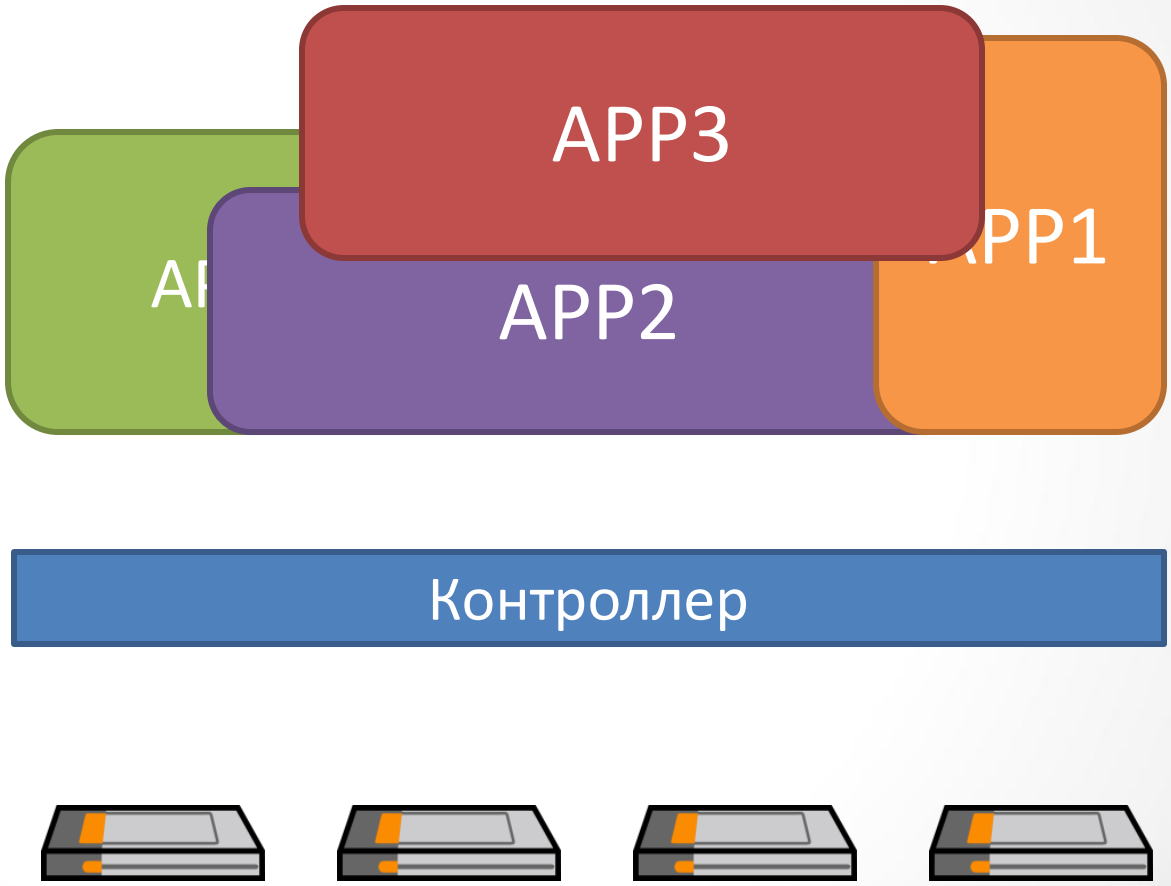
Приложения

API контроллера

Платформа

OpenFlow API

Коммутаторы



Программирование ПКС

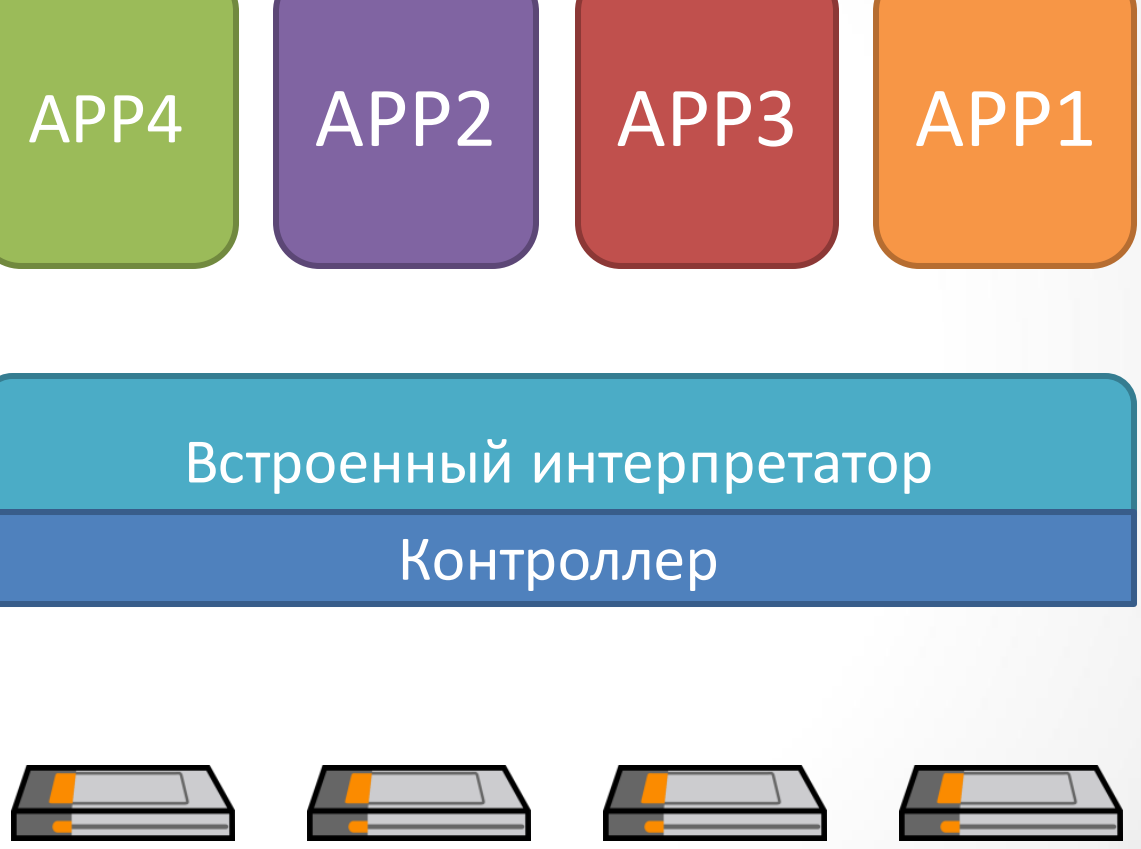
Приложения

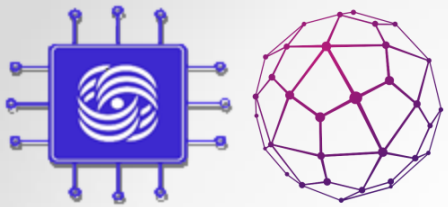
API контроллера

Платформа

OpenFlow API

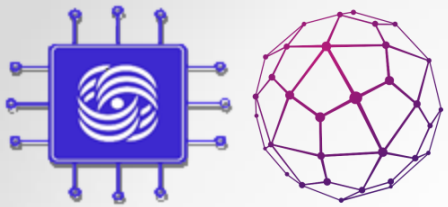
Коммутаторы





Программирование ПКС

```
from pyretic.lib.corelib import *
# отправить пакет на все порты
def main():
        return flood()
# блокировать хост 10.0.0.1
def access_control():
        return ~(match(srcip=’10.0.0.1’) |
                 match(dstip=’10.0.0.1’))
```



Control Plane Verification

Marco Canini, Daniele Venzano et. all

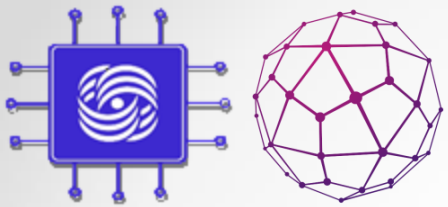
A NICE way to test OpenFlow applications

Строит модель выполнения программы контроллера с учётом состояния всей ПКС

Thomas Ball, Nikolaj Biorner et all.

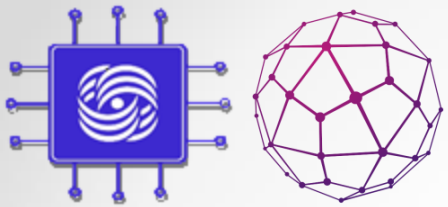
VeriCon: Towards Verifying Controller Programs in Software-Defined Networks

Проверяет программу для всех моделей



Data Plane Verification

- Меньше предположений об управляющих приложениях на контроллере
- Проверяет выполнение политик на уровне проверки правил – метод нечувствителен к *наведённым ошибкам* внутри контроллера
- Естественная интерпретация понятия ***политики маршрутизации***



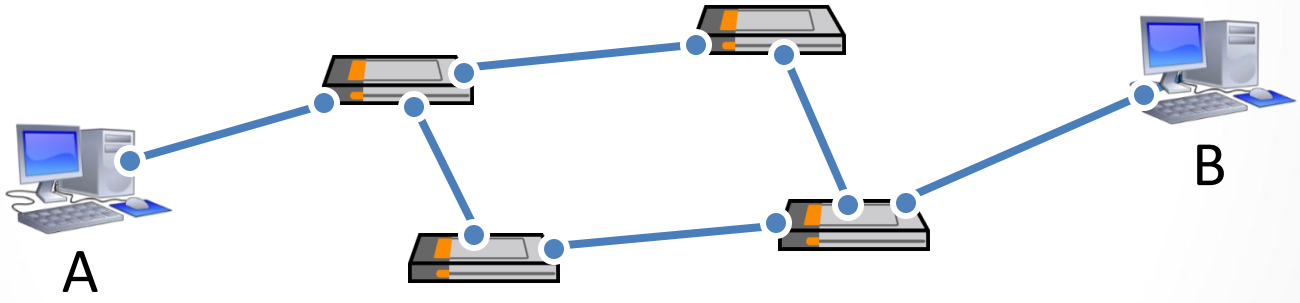
Политики маршрутизации

требования к поведению компьютерной сети

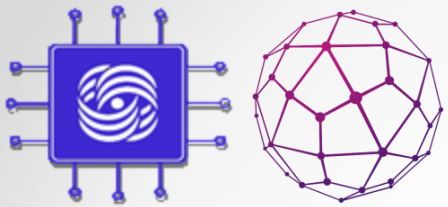
- Внешние потоки не достигают почтового сервера
- Исходящие потоки проходят через средство DPI
- Между каждой парой хостов в офисе есть маршрут
- Сети разных департаментов изолированы
- Все маршруты внутри сети короче шести хопов
- Пакеты не образуют циклов маршрутизации
- Хост A не может подлючиться к хосту B до тех пор, пока хост B не попытался подключиться к хосту A
- Пропускная способность соединения не меньше R, а задержка передачи данных – не превышает T

Анализ свойств отношения маршрутизации

Al-Shaer E., Al-Haj S. FlowChecker: Configuration Analysis and Verification of Federated Openflow Infrastructures // SafeConfig ’10. — Chicago, USA, 2010. — Pp. 37–44.



Kazemian P., Chang M., Zeng H., Varghese G., McKeown N., Whyte S. Real Time Network Policy Checking Using Header Space Analysis // NSDI’13. — Lombard, USA, 2013. — Pp. 99–111.



VERMÓNT
VERifying MONiTor

VERMONT проверяет правила обработки пакетов в таблицах коммутаторов на соответствие формальным спецификациям политик маршрутизации

Необходимо

Выразить требования к поведению сети с помощью нашего языка спецификаций

Одноразовая работа

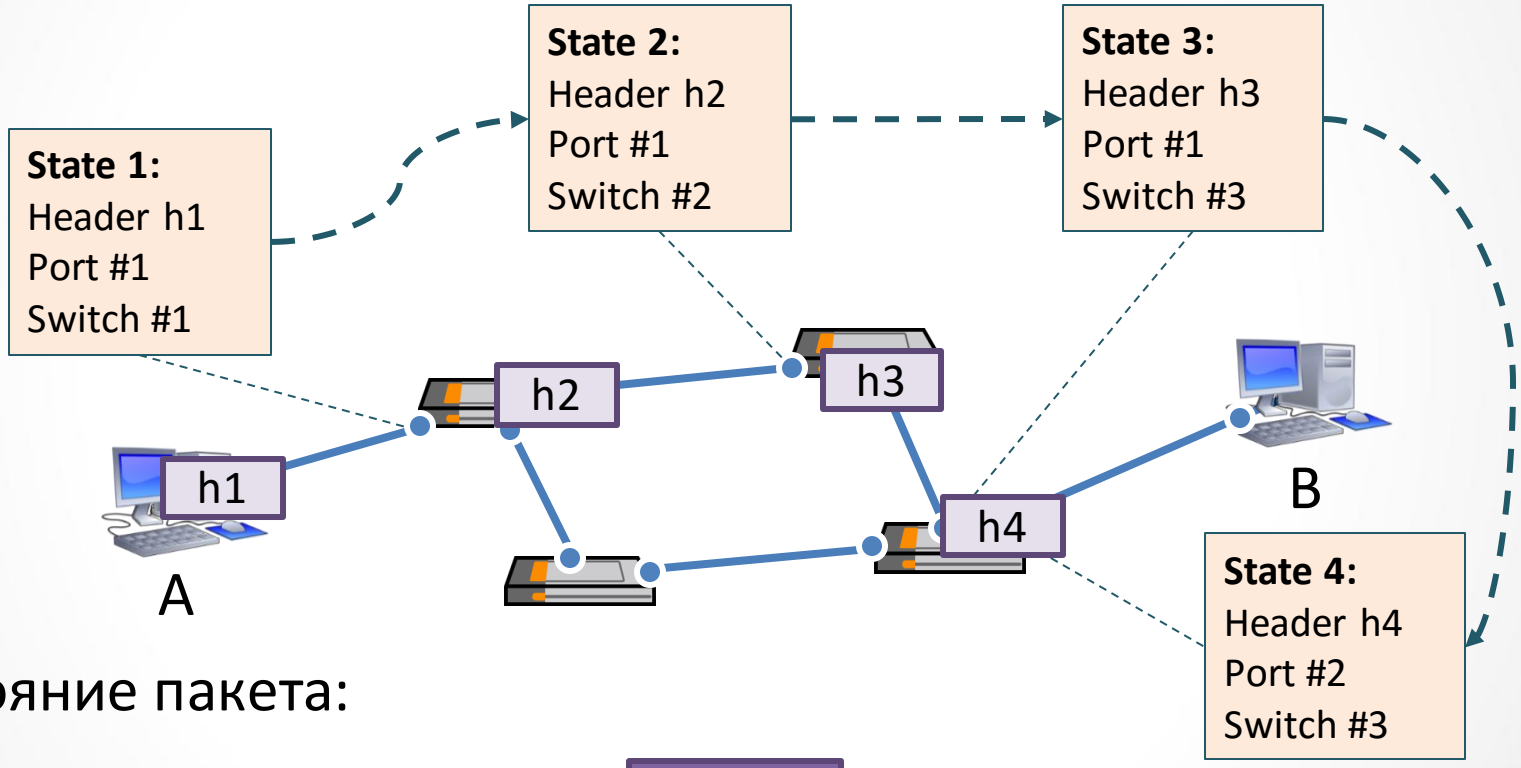
Предоставить топологию и файлы конфигурации коммутационных устройств

Возможна автоматизация

Выгода

- Гарантии работы сети в соответствии с ожиданиями
- Выявление ошибоных конфигураций сети
- Информация о причинах нарушения политики

Анализ свойств отношения маршрутизации



Состояние пакета:

1. Имя коммутатора — Switch — $\{0, 1\}^m$
2. Номер порта — Port — $\{0, 1\}^l$
3. Заголовок пакета — Header — $\{0, 1\}^k$

Реляционная модель сети

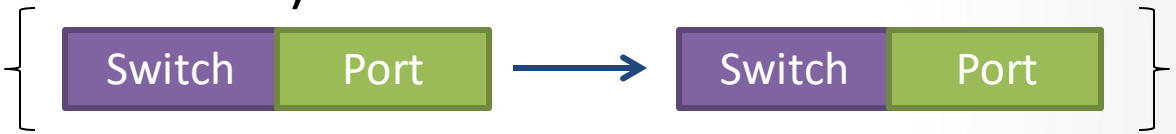
Сеть определяется набором отношений

Входные/выходные порты

(линии связи, топология сети)

Отношение пересылки

(линии связи, топология сети)



Отношения коммутации на узлах

(отдельные правила и таблицы коммутации)

Отношение одношаговой маршрутизации

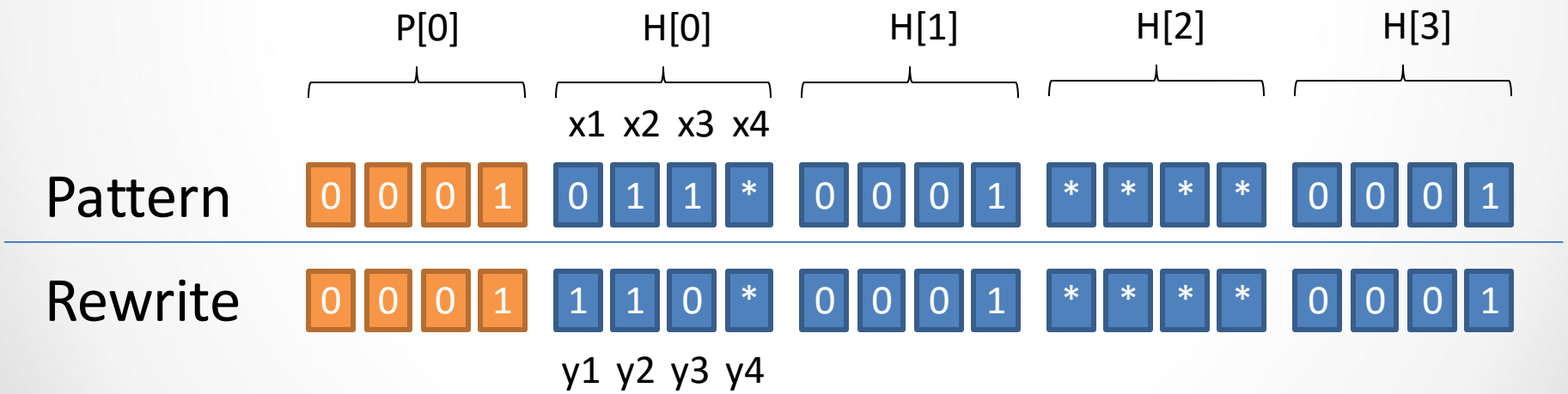
(отдельные коммутаторы, сети коммутаторов)

Реляционная модель сети

Сеть определяется набором отношений

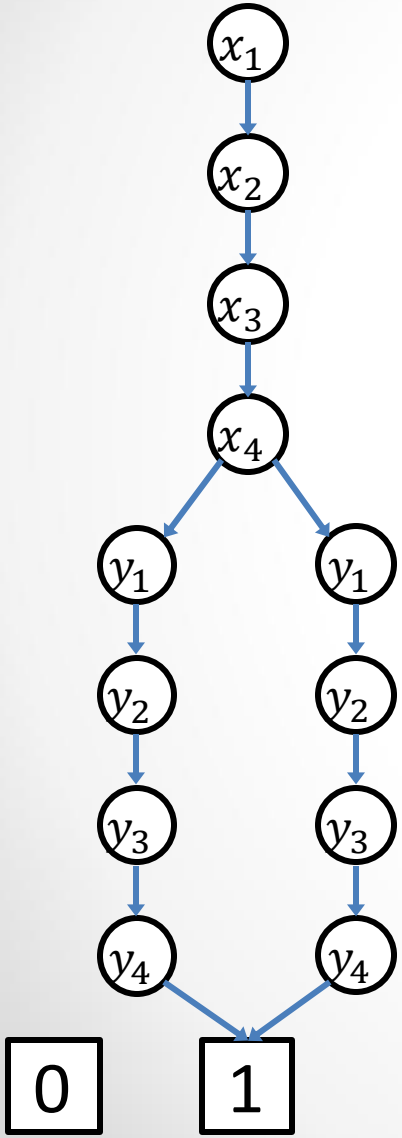
OpenFlow правило моделируется шаблоном Pattern и множеством шаблонов Rewrite

* - пакет подпадает под шаблон Pattern, вне зависимосит от значения бита



* - шаблон Rewrite не изменяет значение бита

Binary Decision Diagram (ROBDD)

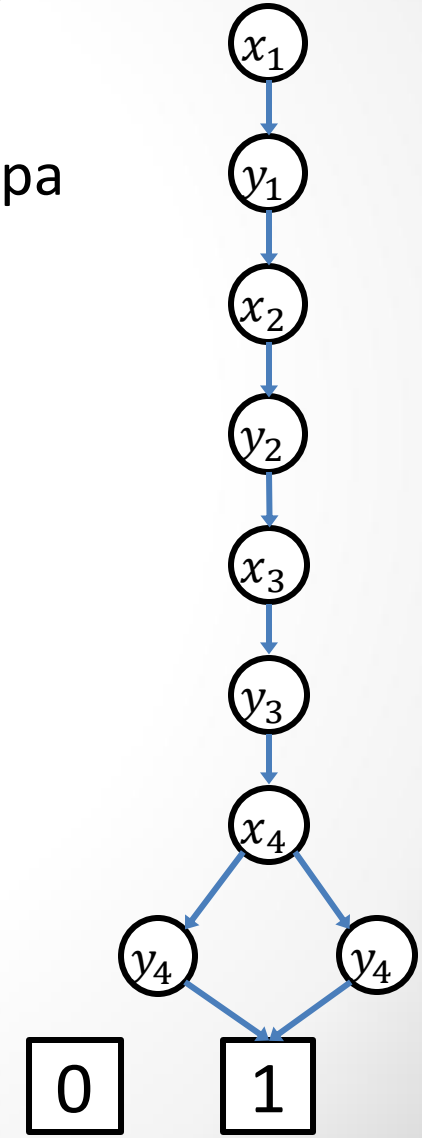


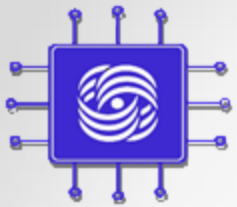
Размер BDD зависит от выбора порядка переменных

x_1	x_2	x_3	x_4
0	1	1	*
1	1	0	*

y_1	y_2	y_3	y_4

$$(\overline{x_1}y_1)(x_2y_2)$$
$$(x_3\overline{y_3})(x_4y_4 \vee \overline{x_4y_4})$$





Построение BDD по состоянию сети

- Строим BDD для каждой пары $\langle Pattern, Rewrite \rangle$
- Строим BDD для каждого правила FlowTable
- Строим BDD для каждого коммутатора
- Строим BDD для всех коммутаторов S
- Строим BDD для топологии T
- Строим композицию S и T
- Получаем отношение R
- Получаем транзитивное замыкание R^*

Язык спецификации

Равенства

Связки

Кванторы

Замыкания

$x[field] = const$	$x[field] = y[field]$	
$\varphi \wedge \varphi$	$\varphi \vee \varphi$	$\neg\varphi$
$\forall x.\varphi$	$\exists x.\varphi$	
ψ^{+}	$\psi^{[i,j]}$	

Правила вычисления замыканий

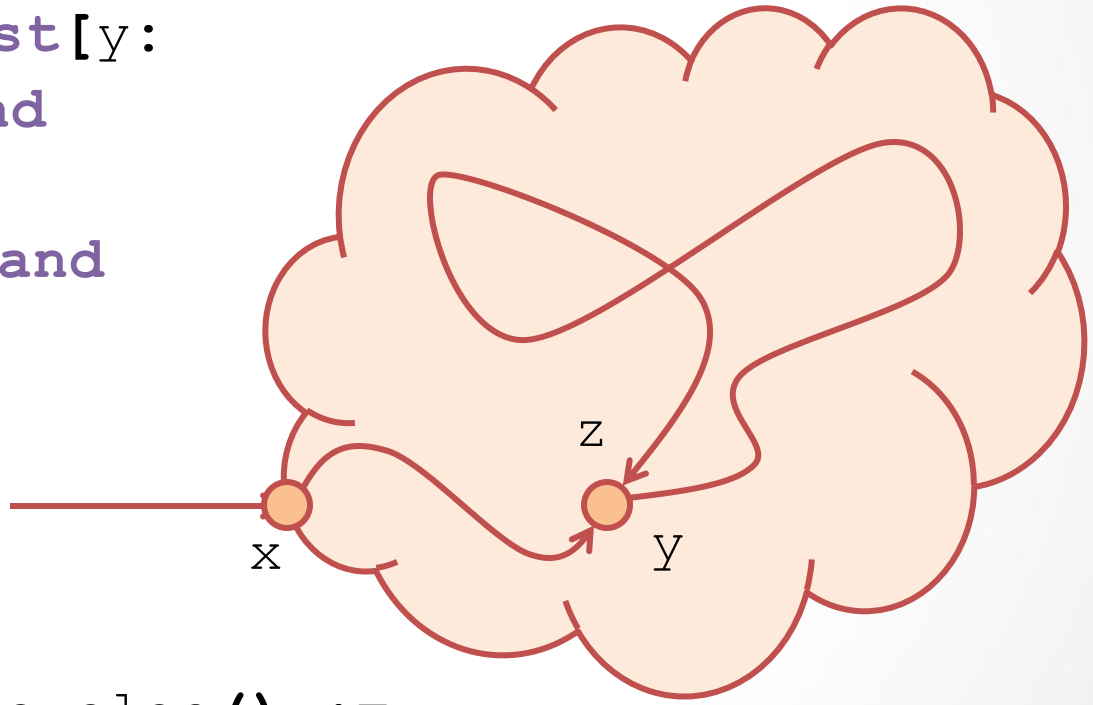
$$\psi^{1}(x,y) = \psi(x,y)$$
$$\psi^{n}(x,y) = \exists z\colon \psi^{n-1}(x,z) \wedge \psi(z,y)$$
$$\psi^{[i,j]}(x,y) = \psi^{i}(x,y) \vee \cdots \vee \psi^{j}(x,y)$$
$$\psi^{+}(x,y) = \psi^{[1,\infty]}(x,y)$$

Отношения, определяющие поведение сети

Входы	$In(x)$
Выходы	$Out(x)$
Шаг коммутации	$R(x,y)$
Отношение достижимости	$R^{+}(x,y)$

Пример: запрет циклов по состоянию пакета

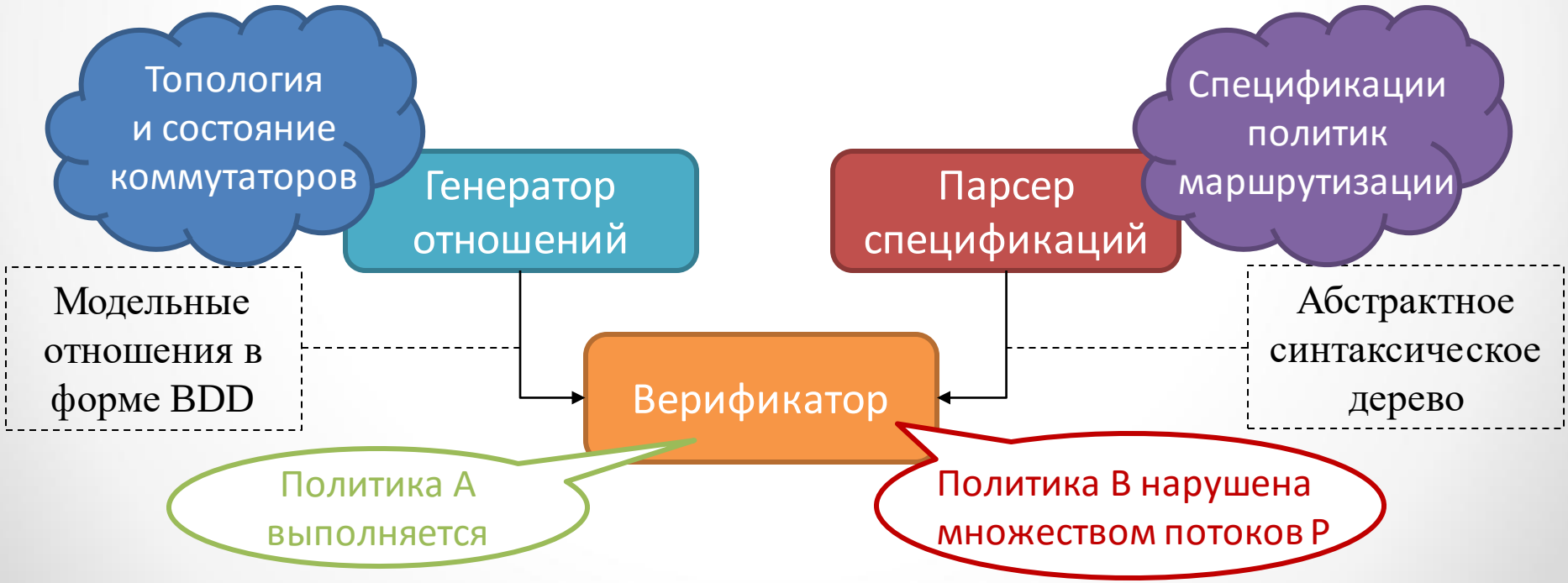
```
aux: lead_to_cycle(x) :=
  In(x) and Exist[y:
    R_tc(x,y) and
    Exist[z:
      R_tc(y,z) and
      y == z
    ]
  ];
```




```
main: no_state_cycles() :=
  Forall[x: not lead_to_cycle(x)];
```

Метод верификации конфигурации сети

Сеть удовлетворяет политике маршрутизации <=> формулы спецификаций этой политики выполняются для отношений, моделирующих указанную сеть



90 Mb файлов с конфигурациями
Топология Fat Tree
16 маршрутизаторов
757000 правил
48 таблиц

Время построения модели, секунды			Свойства R^{+}_{step}		Время верификации ПКС, секунды				
Отношение одношаговой маршрутизации R_{step}	Транзитивное замыкание отношения R_{step}	Общее время работы конструктора	Число вершин дерева OBDD, тыс. шт.	Порядок числа путей дерева OBDD	Циклы по состояниям	Циклы по топологии	Пути длиннее одного скачка	Пути длиннее двух скачков	Пути длинее трёх скачков
1.094	6.294	18.028	642	18	0.043	0.047	0.855	2.013	3.764

Stanford University Backbone Network

Stanford network verification

Проверка требования	Вердикт	Затрачено времени (мс)
Построение модели	-	3043.687
Циклы передачи	YES	166.191
Чёрные дыры	NO	174.845
Маршруты <= 3 хопов	NO	293.522
Маршруты <= 4 хопов	YES	736.015
Вставка правил посл. / парал.	-	100 / 0.3*
Удаление правил посл. / парал.	-	70 / 1*

```
eugene@13inch:~/reps/netver/build$ ./bddg -pm
```

Схема развёртывания

VERMONT моделирует изменения таблиц коммутаторов после применения команды контроллера, и блокирует команду, если она приводит к нарушению политик маршрутизации

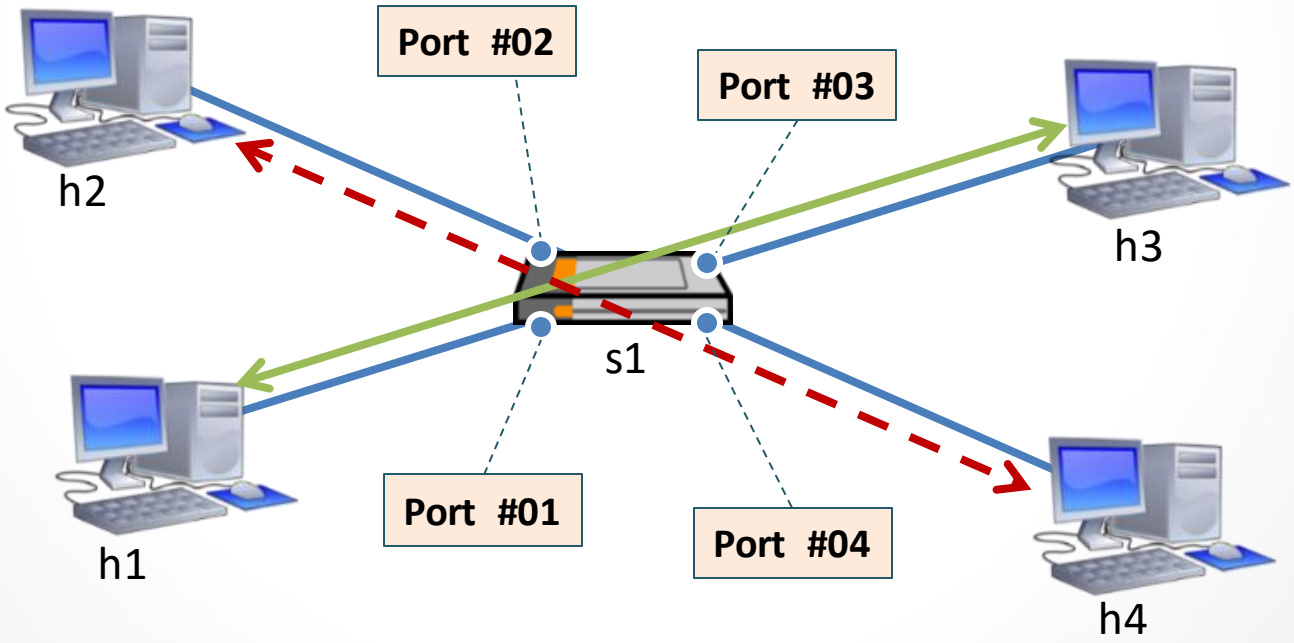
Сравнение с другими средствами

Средство	Построение модели (мс)	Перестроение модели (мс)	Мощность языка	Поддержка OpenFlow
VERMONT 2013	3100	100 - 600	FO[TC]	полная
NetPlumber Stanford University 2013	37000	2 - 1000	CTL	частичная
VeriFlow University of Illinois 2013	>4000	68 - 100	Фиксированный набор свойств	минимальная
AP Verifier University of Texas 2013	1000	0.1	Фиксированный набор свойств	минимальная
Anteater University of Illinois 2011	400000	???	Фиксированный набор свойств	нет
FlowChecker University of North Carolina 2010	1200000	350 - 67000	CTL	полная

Демонстрация VERMONT

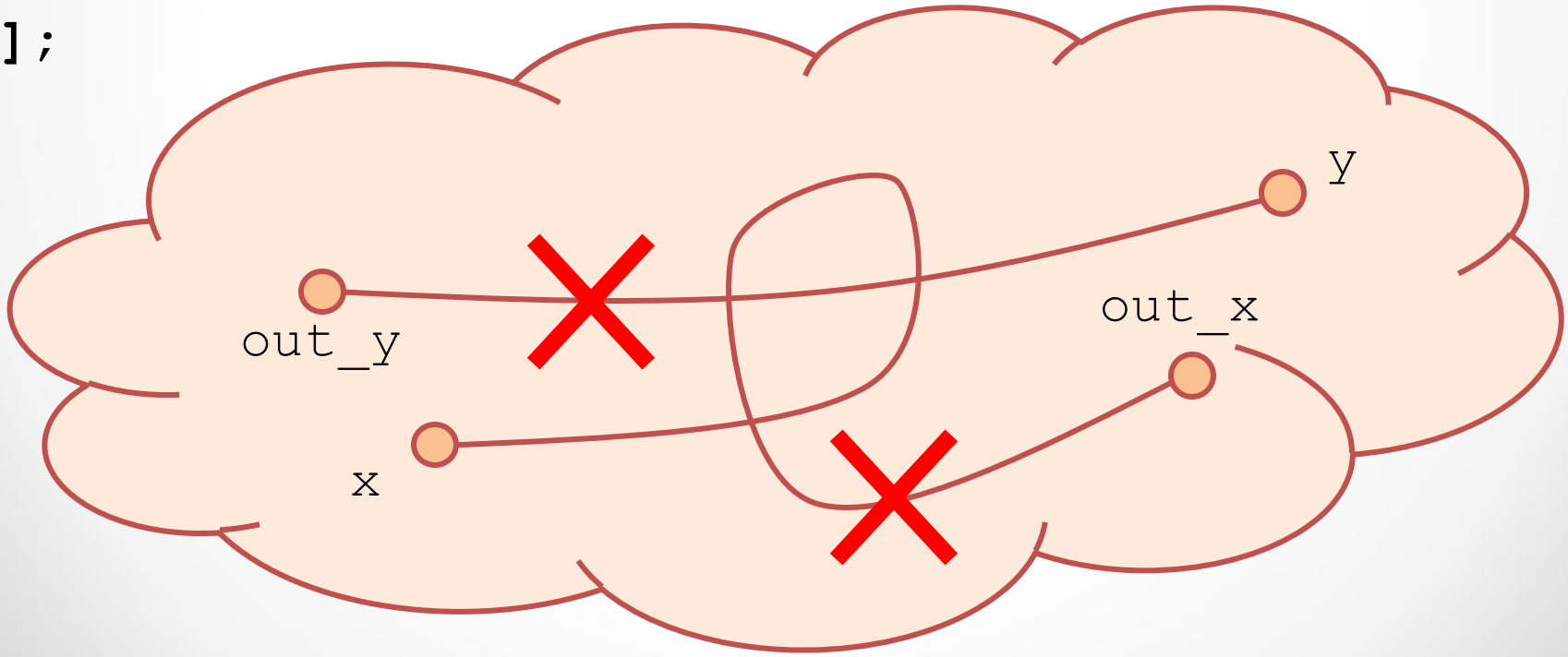
Network disjoint

Коммутатор обслуживает не более двух абонентов одновременно



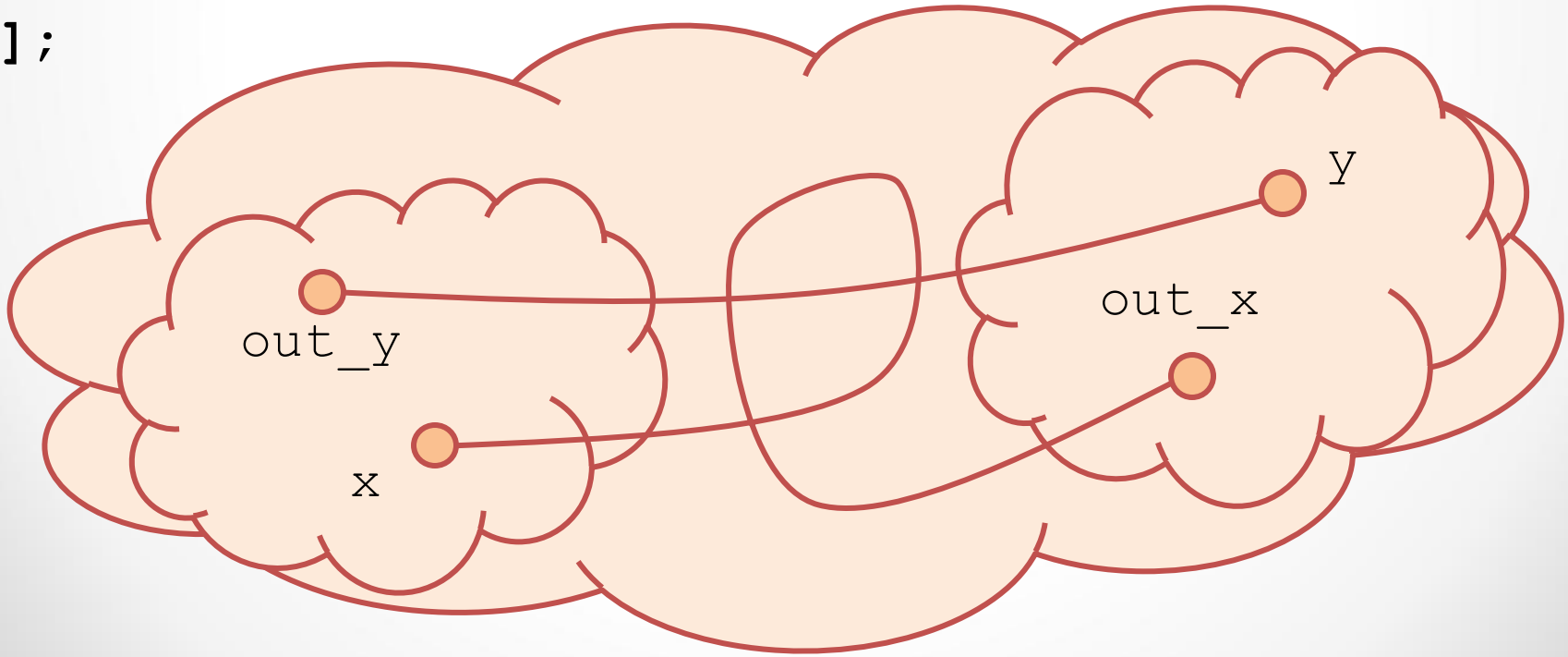
Спецификация политики

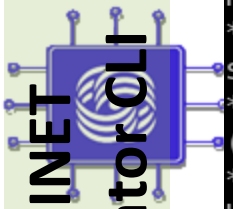
```
main: disjoint() := Forall[x, out_x, y, out_y:
  !R(x, out_x) or !R(y, out_y) or
  x[p] == out_y[p] and out_x[p] == y[p] or
  x[p] == y[p] and out_x[p] == out_y[p]
];
```



Спецификация политики

```
main: disjoint() := Forall[x, out_x, y, out_y:
  !R(x, out_x) or !R(y, out_y) or
  x[p] == out_y[p] and out_x[p] == y[p] or
  x[p] == y[p] and out_x[p] == out_y[p]
];
```





MININET simulator CLI

```
h1 h2 h3 h4
*** Adding switches:
s1
*** Adding links:
(s1, h1) (s1, h2) (s1, h3) (s1, h4)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
*** Starting 1 switches
s1
*** Starting CLI:
mininet>
```

There are more components whose output is not shown:

- SDN controller
- VERMONT Verifier
- VERMONT Feeder

VERMONT proxy CLI

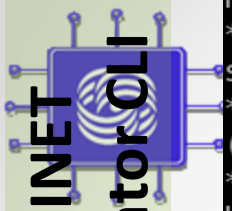
```
Controller addr: 127.0.0.1:6633
VERMONT proxy CLI:
        * help - produce help message
        * get_mode - return code of current proxy server working mode
        * set_mode seamless|mirror|interrupt - set proxy server mode
        * exit - stop proxy server

> adding connection
 attached to thread 139780799772416
 attached to thread 139780791379712
switch with dpid 1 found
```

Switch S1 is already connected to VERMONT proxy

Rules at switch S1

```
NXST_FLOW reply (xid=0x4):
NXST_FLOW reply (xid=0x4):
NXST_FLOW reply (xid=0x4):
NXST_FLOW reply (xid=0x4):
NXST_FLOW reply (xid=0x4):
```

Flow table of switch S1 is currently empty



MININET simulator CLI

```
h1 h2 h3 h4
*** Adding switches:
s1
*** Adding links:
(s1, h1) (s1, h2) (s1, h3) (s1, h4)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
*** Starting 1 switches
s1
*** Starting CLI:
mininet>
```

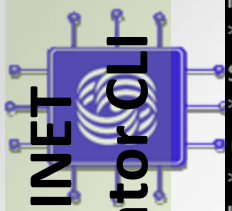
VERMONT proxy CLI

```
Controller addr: 127.0.0.1:6633
VERMONT proxy CLI:
        * help - produce help message
        * get_mode - return code of current proxy server working mode
        * set_mode seamless|mirror|interrupt - set proxy server mode
        * exit - stop proxy server

> adding connection
 attached to thread 139780799772416
 attached to thread 139780791379712
switch with dpid 1 found
s
```

Setting VERMONT proxy to interrupt mode

Rules at switch S1

```
NXST_FLOW reply (xid=0x4):
NXST_FLOW reply (xid=0x4):
NXST_FLOW reply (xid=0x4):
NXST_FLOW reply (xid=0x4):
NXST_FLOW reply (xid=0x4):
```



MININET simulator CLI

```
h1 h2 h3 h4
*** Adding switches:
s1
*** Adding links:
(s1, h1) (s1, h2) (s1, h3) (s1, h4)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
*** Starting 1 switches
s1
*** Starting CLI:
mininet>
```

Host h1 starts to ping host h3

VERMONT proxy CLI

```
        * get_mode - return code of current proxy server working mode
        * set_mode seamless|mirror|interrupt - set proxy server mode
        * exit - stop proxy server

> adding connection
 attached to thread 139780799772416
 attached to thread 139780791379712
switch with dpid 1 found
set_mode interrupt
connection to verifier established
Connected to verifier (127.0.0.1:3366)
>
```

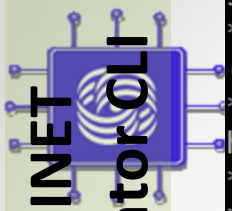
Controller tries to install the rules to transmit ping packets

Proxy interrupts command from the controller and sends them to Verifier

Rules at switch S1

```
NXST_FLOW reply (xid=0x4):
NXST_FLOW reply (xid=0x4):
NXST_FLOW reply (xid=0x4):
NXST_FLOW reply (xid=0x4):
NXST_FLOW reply (xid=0x4):
```

Verifier create an updated model of the data plane and checks them against a set of PFPs



MININET simulator CLI

```
s1
*** Adding links:
(s1, h1) (s1, h2) (s1, h3) (s1, h4)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
*** Starting 1 switches
s1
*** Starting CLI:
mininet> h1 ping h3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
```

First packet of the flow uses slow path

VERMONT proxy CLI

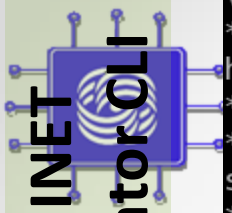
```
        * get_mode - return code of current proxy server working mode
        * set_mode seamless|mirror|interrupt - set proxy server mode
        * exit - stop proxy server

> adding connection
 attached to thread 139780799772416
 attached to thread 139780791379712
switch with dpid 1 found
set_mode interrupt
connection to verifier established
Connected to verifier (127.0.0.1:3366)
>
```

Rules at switch S1

```
NXST_FLOW reply (xid=0x4):
NXST_FLOW reply (xid=0x4):
NXST_FLOW reply (xid=0x4):
NXST_FLOW reply (xid=0x4):
NXST_FLOW reply (xid=0x4):
```

Proxy delivers verified commands to the switch

Switch table contains rules to transmit packets between host h1 and host h3



MININET simulator CLI

```
(s1, h1) (s1, h2) (s1, h3) (s1, h4)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
*** Starting 1 switches
s1
*** Starting CLI:
mininet> h1 ping h3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=85.1 ms
64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=0.132 ms
```

Subsidiary packets use fast path

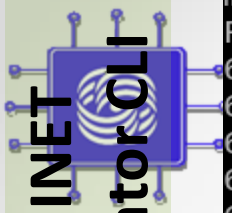
VERMONT proxy CLI

```
        * get_mode - return code of current proxy server working mode
        * set_mode seamless|mirror|interrupt - set proxy server mode
        * exit - stop proxy server

> adding connection
 attached to thread 139780799772416
 attached to thread 139780791379712
switch with dpid 1 found
set_mode interrupt
connection to verifier established
Connected to verifier (127.0.0.1:3366)
>
```

Rules have an idle timeout and will expire in 5 seconds

Rules at switch S1

```
NXST_FLOW reply (xid=0x4):

NXST_FLOW reply (xid=0x4):
 cookie=0x20000000000000, duration=0.135s, table=0, n_packets=1, n_bytes=98, idle_tim
eout=5, idle_age=0, priority=0,in_port=3,vlan_tci=0x0000,dl_src=00:00:00:00:00:03,dl_
dst=00:00:00:00:00:01 actions=output:1
 cookie=0x20000000000000, duration=0.094s, table=0, n_packets=0, n_bytes=0, idle_time
out=5, idle_age=0, priority=0,in_port=1,vlan_tci=0x0000,dl_src=00:00:00:00:00:01,dl_d
st=00:00:00:00:00:03 actions=output:3
```



MININET simulator CLI

```
mininet> h1 ping h3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=85.1 ms
64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=0.132 ms
64 bytes from 10.0.0.3: icmp_seq=3 ttl=64 time=0.070 ms
64 bytes from 10.0.0.3: icmp_seq=4 ttl=64 time=0.068 ms
64 bytes from 10.0.0.3: icmp_seq=5 ttl=64 time=0.068 ms
^C
--- 10.0.0.3 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4000ms
rtt min/avg/max/mdev = 0.068/17.103/85.177/34.037 ms
mininet>
```

Host h2 starts to ping host h4

VERMONT proxy CLI

```
        * get_mode - return code of current proxy server working mode
        * set_mode seamless|mirror|interrupt - set proxy server mode
        * exit - stop proxy server

> adding connection
 attached to thread 139780799772416
 attached to thread 139780791379712
switch with dpid 1 found
set_mode interrupt
connection to verifier established
Connected to verifier (127.0.0.1:3366)
>
```

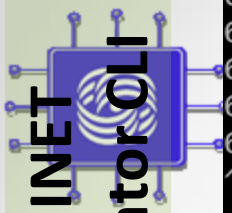
Controller tries to install the rules to transmit ping packets

Proxy interrupts command from the controller and sends them to Verifier

Rules at switch S1

```
meout=5, idle_age=0, priority=0,in_port=1,vlan                    :01,dl
_dst=00:00:00:00:00:03 actions=output:3
-------------------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
 cookie=0x20000000000000, duration=4.175s, table=0, n_packets=3, n_bytes=490, idle_ti
meout=5, idle_age=0, priority=0,in_port=3,vlan_tci=0x0000,dl_src=00:00:00:00:00:03,dl
_dst=00:00:00:00:00:01 actions=output:1
 cookie=0x20000000000000, duration=4.134s, table=0, n_packets=4, n_bytes=392, idle_ti
meout=5, idle_age=0, priority=0,in_port=1,vlan_tci=0x0000,dl_src=00:00:00:00:00:01,dl
_dst=00:00:00:00:00:03 actions=output:3
-------------------------------------------------------------------------
```

Verifier create an updated model of the data plane and checks them against a set of PFPs



MININET simulator CLI

```
64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=85.1 ms
64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=0.132 ms
64 bytes from 10.0.0.3: icmp_seq=3 ttl=64 time=0.070 ms
64 bytes from 10.0.0.3: icmp_seq=4 ttl=64 time=0.068 ms
64 bytes from 10.0.0.3: icmp_seq=5 ttl=64 time=0.068 ms
^C
--- 10.0.0.3 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4000ms
rtt min/avg/max/mdev = 0.068/17.103/85.177/34.037 ms
mininet> h2 ping h4
PING 10.0.0.4 (10.0.0.4) 56(84) bytes of data.
```

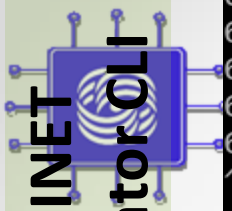
VERMONT proxy CLI

```
        * get_mode - return code of current proxy server working mode
        * set_mode seamless|mirror|interrupt - set proxy server mode
        * exit - stop proxy server

> adding connection
 attached to thread 139780799772416
 attached to thread 139780791379712
switch with dpid 1 found
set_mode interrupt
connection to verifier established
Connected to verifier (127.0.0.1:3366)
>
```

Proxy drops unsafe commands and notifies the controller

Rules at switch S1

```
meout=5, idle_age=0, priority=0,in_port=1,vlan_tci=0x0000,dl_src=00:00:00:00:00:01,dl
_dst=00:00:00:00:00:03 actions=output:3
-------------------------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
 cookie=0x20000000000000, duration=6.197s, table=0, n_packets=6, n_bytes=532, idle_ti
meout=5, idle_age=1, priority=0,in_port=3,vlan_tci=0x0000,dl_src=00:00:00:00:00:03,dl
_dst=00:00:00:00:00:01 actions=output:1
 cookie=0x20000000000000, duration=6.156s, table=0, n_packets=5, n_bytes=434, idle_ti
meout=5, idle_age=1, priority=0,in_port=1,vlan_tci=0x0000,dl_src=00:00:00:00:00:01,dl
_dst=00:00:00:00:00:03 actions=output:3
-------------------------------------------------------------------------------
```



MININET simulator CLI

```
64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=85.1 ms
64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=0.132 ms
64 bytes from 10.0.0.3: icmp_seq=3 ttl=64 time=0.070 ms
64 bytes from 10.0.0.3: icmp_seq=4 ttl=64 time=0.068 ms
64 bytes from 10.0.0.3: icmp_seq=5 ttl=64 time=0.068 ms
^C
--- 10.0.0.3 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4000ms
rtt min/avg/max/mdev = 0.068/17.103/85.177/34.037 ms
mininet> h2 ping h4
PING 10.0.0.4 (10.0.0.4) 56(84) bytes of data.
```

VERMONT proxy CLI

```
        * exit - stop proxy server

> adding connection
 attached to thread 139780799772416
 attached to thread 139780791379712
switch with dpid 1 found
set_mode interrupt
connection to verifier established
Connected to verifier (127.0.0.1:3366)
> command blocked
command blocked
```

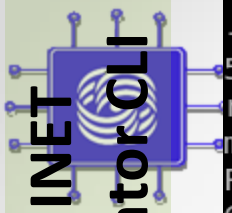
Why does ping work?

Packets are delivered through the control plane

We can block them!
But do we really want to?

Rules at switch S1

```
meout=5, idle_age=0, priority=0,in_port=1,vlan_tci=0x0000,dl_src=00:00:00:00:00:01,dl
_dst=00:00:00:00:00:03 actions=output:3
-------------------------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
 cookie=0x20000000000000, duration=6.197s, table=0, n_packets=6, n_bytes=532, idle_ti
meout=5, idle_age=1, priority=0,in_port=3,vlan_tci=0x0000,dl_src=00:00:00:00:00:03,dl
_dst=00:00:00:00:00:01 actions=output:1
 cookie=0x20000000000000, duration=6.156s, table=0, n_packets=5, n_bytes=434, idle_ti
meout=5, idle_age=1, priority=0,in_port=1,vlan_tci=0x0000,dl_src=00:00:00:00:00:01,dl
_dst=00:00:00:00:00:03 actions=output:3
-------------------------------------------------------------------------------
```



MININET simulator CLI

```
^C
--- 10.0.0.3 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4000ms
rtt min/avg/max/mdev = 0.068/17.103/85.177/
mininet> h2 ping h4
PING 10.0.0.4 (10.0.0.4) 56(84) bytes of data.
64 bytes from 10.0.0.4: icmp_seq=1 ttl=64 time=91.5 ms
64 bytes from 10.0.0.4: icmp_seq=2 ttl=64 time=45.1 ms
64 bytes from 10.0.0.4: icmp_seq=3 ttl=64 time=46.5 ms
64 bytes from 10.0.0.4: icmp_seq=4 ttl=64 time=47.5 ms
64 bytes from 10.0.0.4: icmp_seq=5 ttl=64 time=45.5 ms
```

Packets start to use fast path

VERMONT proxy CLI

```
connection to verifier established
Connected to verifier (127.0.0.1:3366)
> command blocked
command blocked
command blocked
command blocked
command blocked
command blocked
command blocked
command blocked
command blocked
```

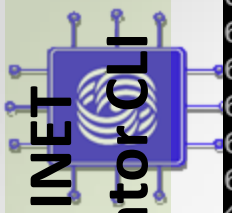
Old rules have been expired

Controller is allowed to install new rules

Rules at switch S1

```
imeout=5, idle_age=5, priority=0,in_port=1,vla                    0:01,c
l_dst=00:00:00:00:00:03 actions=output:3
-----------------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
 cookie=0x20000000000000, duration=0.629s, tab                    idle_tim
eout=5, idle_age=0, priority=0,in_port=4,vlan_tci=0x0000,dl_src=00:00:00:00:00:04,dl_
dst=00:00:00:00:00:02 actions=output:2
 cookie=0x20000000000000, duration=0.633s, table=0, n_packets=1, n_bytes=98, idle_tim
eout=5, idle_age=0, priority=0,in_port=2,vlan_tci=0x0000,dl_src=00:00:00:00:00:02,dl_
dst=00:00:00:00:00:04 actions=output:4
-----------------------------------------------------------------------
```

Switch table contains rules to transmit packets between host h2 and host h4



MININET simulator CLI

```
64 bytes from 10.0.0.4: icmp_seq=5 ttl=64 time=45.5 ms
64 bytes from 10.0.0.4: icmp_seq=6 ttl=64 time=0.159 ms
64 bytes from 10.0.0.4: icmp_seq=7 ttl=64 time=0.042 ms
64 bytes from 10.0.0.4: icmp_seq=8 ttl=64 time=0.042 ms
64 bytes from 10.0.0.4: icmp_seq=9 ttl=64 time=0.037 ms
64 bytes from 10.0.0.4: icmp_seq=10 ttl=64 time=0.055 ms
^C
--- 10.0.0.4 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9007ms
rtt min/avg/max/mdev = 0.037/27.672/91.542/30.443 ms

mininet>
```

Packets use slow path

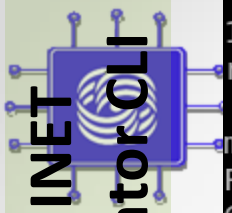
VERMONT proxy CLI

```
connection to verifier established
Connected to verifier (127.0.0.1:3366)
> command blocked
command blocked
command blocked
command blocked
command blocked
command blocked
command blocked
command blocked
command blocked
```

Rules to transmit packets from host h1 to host h3 violate forwarding policy

Rules at switch S1

```
meout=5, idle_age=0, priority=0,in_port=2,vlan_tci=0x0000,dl_src=00:00:00:00:00:02,dl
_dst=00:00:00:00:00:04 actions=output:4
-----------------------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
 cookie=0x20000000000000, duration=4.671s, table=0, n_packets=6, n_bytes=532, idle_ti
meout=5, idle_age=0, priority=0,in_port=4,vlan_tci=0x0000,dl_src=00:00:00:00:00:04,dl
_dst=00:00:00:00:00:02 actions=output:2
 cookie=0x20000000000000, duration=4.675s, table=0, n_packets=6, n_bytes=532, idle_ti
meout=5, idle_age=0, priority=0,in_port=2,vlan_tci=0x0000,dl_src=00:00:00:00:00:02,dl
_dst=00:00:00:00:00:04 actions=output:4
-----------------------------------------------------------------------------
```



MININET simulator CLI

```
--- 10.0.0.4 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9007ms
rtt min/avg/max/mdev = 0.037/27.672/91.542/30.443 ms

mininet> h1 ping h3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=45.9 ms
64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=50.5 ms
64 bytes from 10.0.0.3: icmp_seq=3 ttl=64 time=47.1 ms
64 bytes from 10.0.0.3: icmp_seq=4 ttl=64 time=41.8 ms
64 bytes from 10.0.0.3: icmp_seq=5 ttl=64 time=45.8 ms
```

Packets start to use fast path

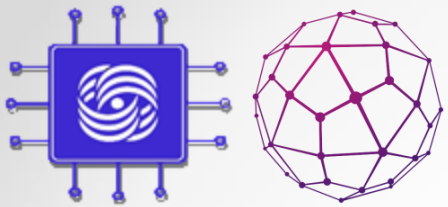
VERMONT proxy CLI

```
command blocked
command blocked
command blocked
command blocked
command blocked
command blocked
command blocked
command blocked
command blocked
command blocked
command blocked
```

Rules at switch S1

```
meout=5, idle_age=4, priority=0,in_port=2,vlan                    :02,dl
_dst=00:00:00:00:00:04 actions=output:4
-------------------------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
 cookie=0x20000000000000, duration=0.570s, table=0, n_packets=1, n_bytes=98, idle_tim
eout=5, idle_age=0, priority=0,in_port=3,vlan_tci=0x0000,dl_src=00:00:00:00:00:03,dl_
dst=00:00:00:00:00:01 actions=output:1
 cookie=0x20000000000000, duration=0.573s, table=0, n_packets=1, n_bytes=98, idle_tim
eout=5, idle_age=0, priority=0,in_port=1,vlan_tci=0x0000,dl_src=00:00:00:00:00:01,dl_
dst=00:00:00:00:00:03 actions=output:3
-------------------------------------------------------------------------------
```

Switch table contains rules to transmit packets between host h1 and host h3



Проблема консистентного обновления конфигурации

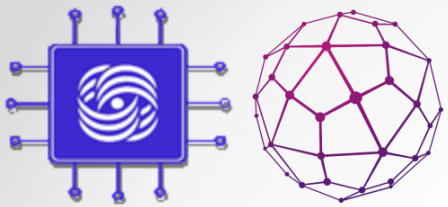
- ***Конфигурация сети*** C задаёт привязку правил к коммутаторам и топологию сети
- Отношения In_C и R_C для конфигурации C определяет ***пути*** $path$ передачи пакетов

$$s_0 \in In_C$$

$$(s_i, s_{i+1}) \in R_C$$

$$path = s_0, s_1, \dots, s_i, s_{i+1}, \dots$$

- $Path(C)$ – множество всех путей передачи пакетов для конфигурации C



Проблема консистентного обновления конфигурации

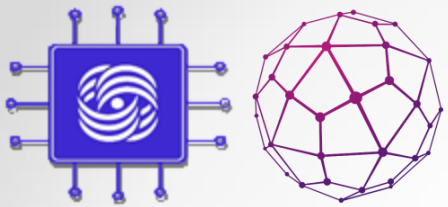
- com – ***команда реконфигурации*** сети

  добавление, удаление или модификация правила маршрутизации

- $com(C)$ - конфигурация, полученнная в результате ***применения команды*** com к конфигурации C

- Если $\alpha = com_1, \dots, com_k$, тогда

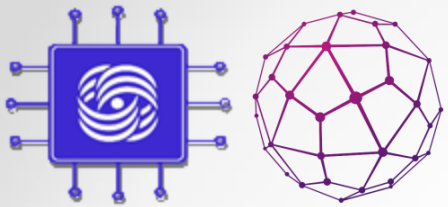
$$\alpha(C) = com_k(\dots, com_1(C) \dots)$$



Проблема консистентного обновления конфигурации

- На множестве команд реконфигурации Com введено ***отношение частичного порядка*** $\prec$: если $com' \prec com''$, то com'' выполняется только после выполнения com'

- ***Пакет реконфигурации*** – множество команд реконфигурации, дополненное отношением частичного порядка $(Com, \prec)$.



Проблема консистентного обновления конфигурации

Входные данные:

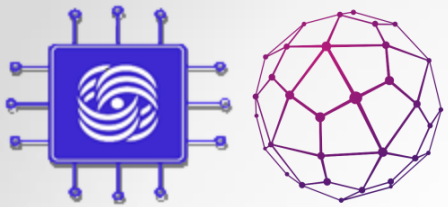
- Начальная конфигурация сети C_0
- Требования корректности и безопасности

пост-условие Φ, ***инвариант*** Ψ.

Требования к решению:

- Такой пакет реконфигурации $(U, \prec)$, для любой линеаризации которого α_U выполнено:

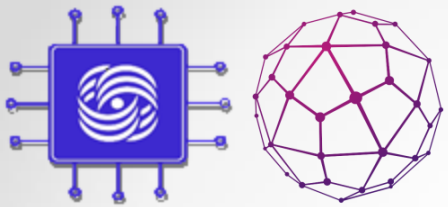
1. $\alpha_U(C_0) \vDash \Phi$
2. $\forall \alpha' \; (\; \alpha_U = \alpha' \alpha'' \; \Rightarrow \alpha'(C_0) \vDash \Psi \;)$



Синтез сетевых конфигураций

Построить такую конфигурацию C, которая удовлетворяла бы заданному постусловию Φ

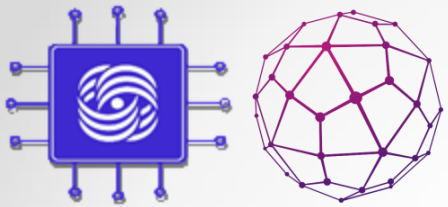
- A. Noyes, T. Warszawski, P. Cernyand, N. Foster. *Toward Synthesis of Network Updates.* 2-nd Workshop on Synthesis (CAV-2013), 2013, Saint Petersburg, Russia



Глобально-консистентное обновление сети

- Post-condition $\Psi(\mathrm{X})$:
- $X = C$
- Invariant $\Phi(\mathrm{X})$:
- $\forall\ s\ (In(s) \rightarrow\ ((Path(X,s) \subseteq\ Path(C)) \vee (Path(X,s) \subseteq\ Path(C_0))))$

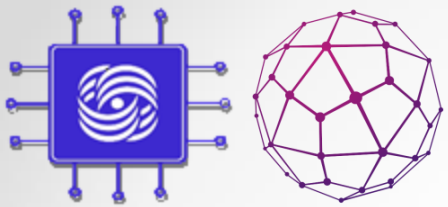
- M. Reitblatt, N. Foster, J. Rexford, D. Walker.
- *Consistent updates for software-defined networks: change you can believe in!*
- *HotNets, v. 7, 2011.*



Локально-консистентное обновление сети

- Post-condition $\Psi(X)$:
- $Path(X) = (Path(C_0) \setminus \{path_0\}) \cup \{path_1\}$
- Invariant $\Phi(X)$:
- $Path(C_0) \setminus \{path_0\} \subseteq Path(X) \subseteq Path(C_0) \cup \{path_1\}$

S. Raza, Y. Zhu, C.-N. Chu S. Raza, Y. Zhu, C.-N. Chuah.
Graceful network state migrations.
IEEE/ACM Transactions on Networking, v. 19, N 4, 2011.



Восстановление сети

- Пост-условие $\Psi(X)$:
- $X = C$
- Инвариант $\Phi(X)$:
- $Path(C_0) \cap Path(C) \subseteq Path(X) \subseteq Path(C_0) \cup Path(C)$

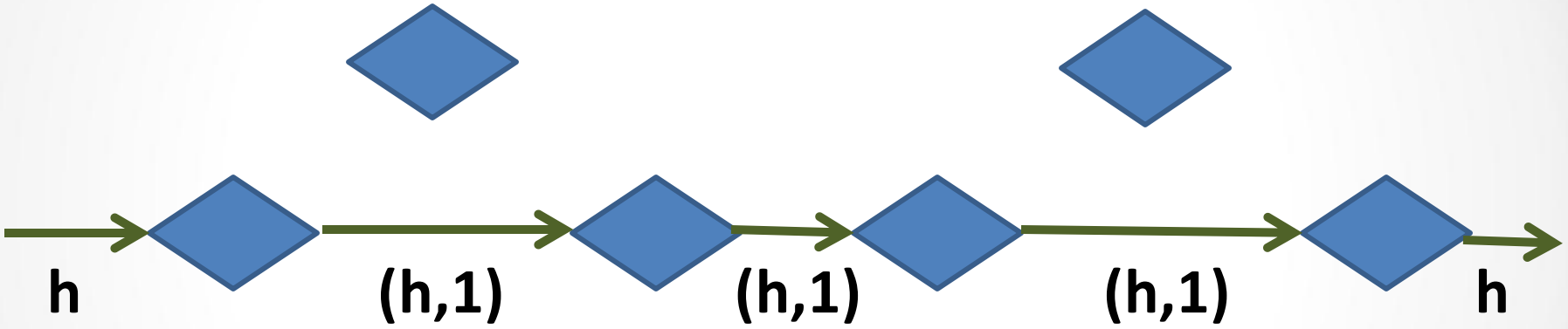
- Конфигурация C перешла в C_0 в результате удаления части правил
- Цель – восстановить C_0 из конфигурации C

Оптимизация таблицы потоков

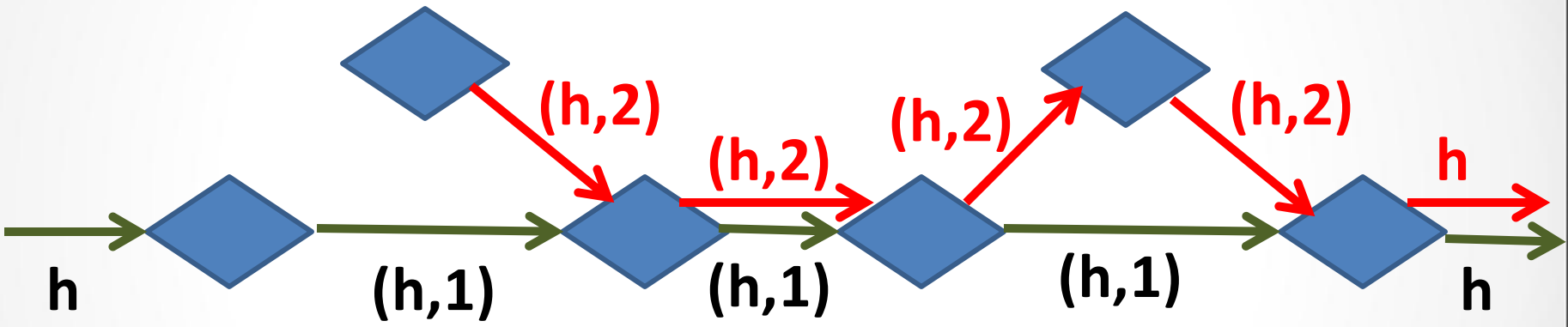
- Пост-условие $\Psi(X)$:
- $(Path(X) = Path(C)) \wedge$
- $\forall Y(Path(X) = Path(C) \rightarrow f(X) \leq f(Y))$.
- Инвариант $\Phi(X)$:
- $Path(X) = Path(C)$,

- K. Kogan, S. Nikolenko, W. Culhane, P. Eugster, E. Ruan. *Towards efficient implementation of packet classifiers.* Proc. of the 2-d Workshop on Hot Topics in SDN, 2013.

Обновление сети с помощью тегов



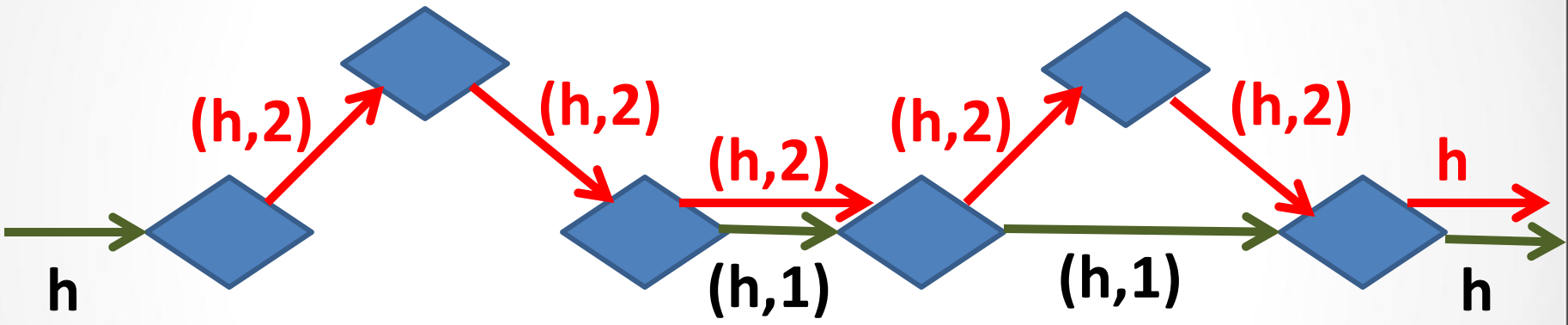
Обновление сети с помощью тегов



3-х фазовый алгоритм обновления

1: добавление новых правил

Обновление сети с помощью тегов

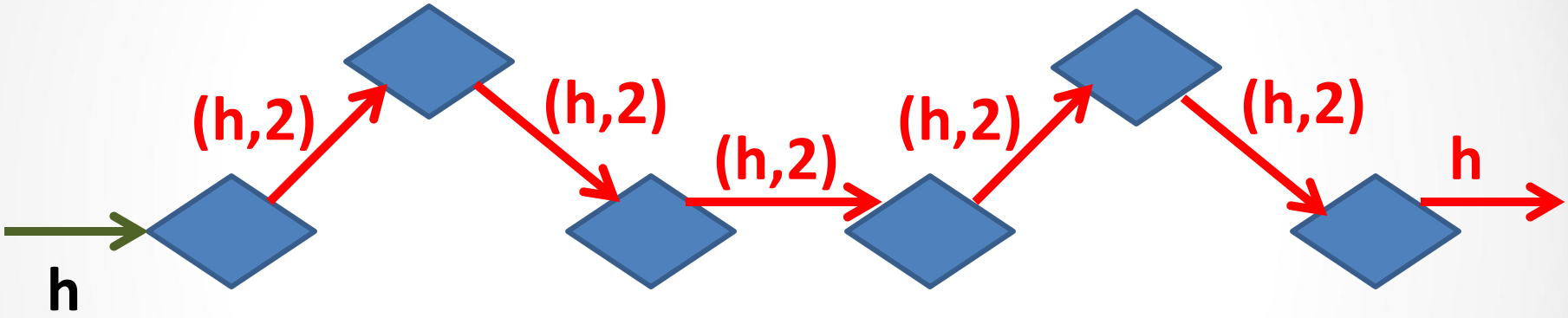


3-х фазовый алгоритм обновления

1: добавление новых правил

2: переключение потоков

Обновление сети с помощью тегов

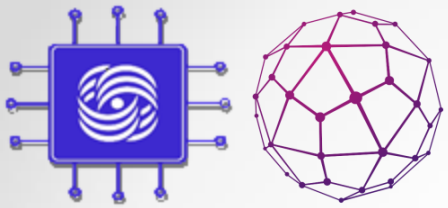


3-х фазовый алгоритм обновления

1: добавление новых правил

2: переключение потоков

3: удаление устаревших правил



Заключение