Performance and Security Evaluation of SDN Networks in OMNeT++/INET

Marco Tiloca, Alexandra Stagkopoulou, Gianluca Dini

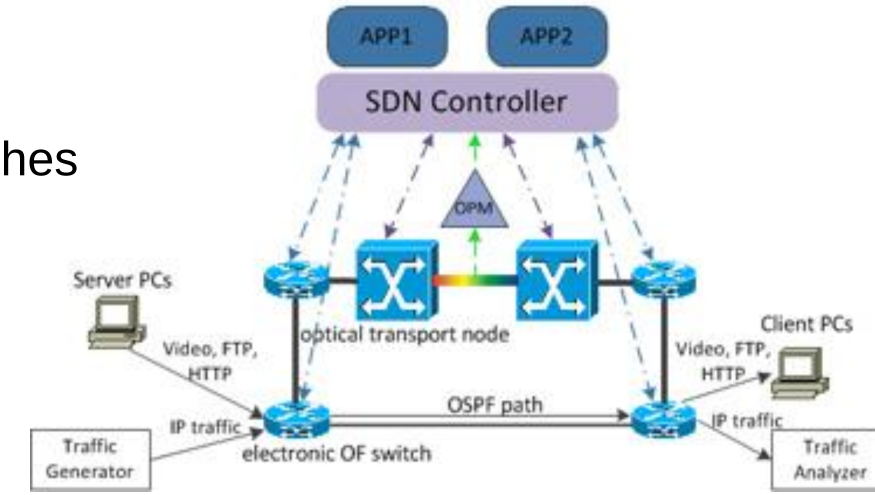
SWEDISH ICT SICS

KTH VETENSKAP OCH KONST

UNIVERSITÀ DI PISA

Software Defined Networking - Overview

- Key concepts
  - Separation of *Control plane* and *Data plane*
  - Centralized SDN controller and simple Switches

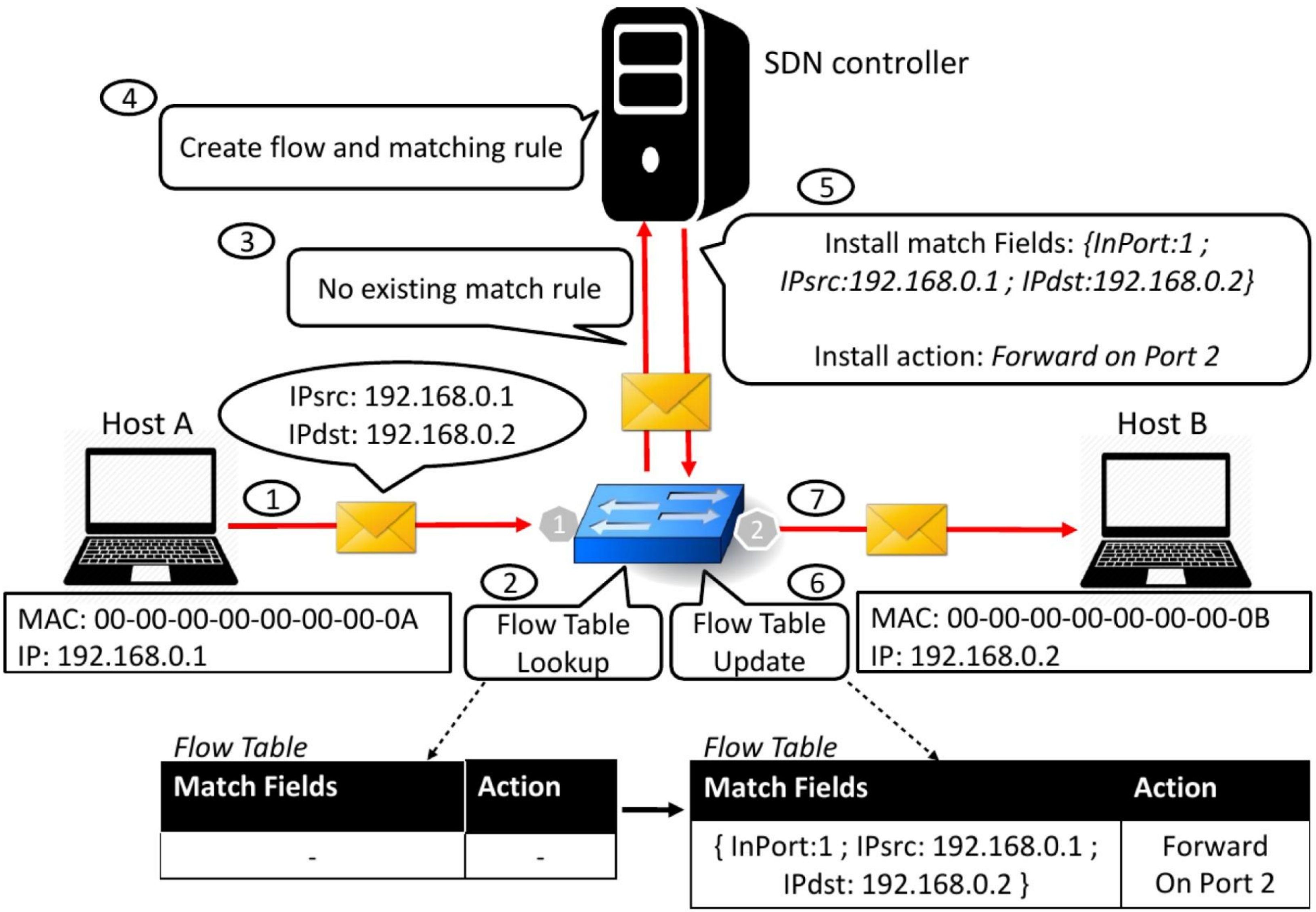


- Control plane
  - Management of routes and network traffic
  - Establishment of routes and *flows*

- Data plane
  - Forwarding of network packets
  - Based on flows and packet-matching rules

- *OpenFlow* as de-facto standard
  - Control messages and APIs for controllers and switches
  - Interoperability among different platforms and vendors

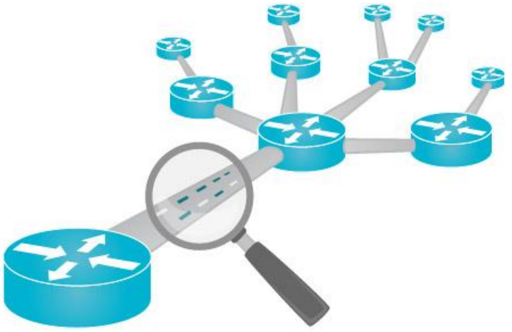
Software Defined Networking - Overview



Need for evaluation tools

- Quantitative assessment of SDN systems
  - At design time (before deployment!)
  - Avoid practically infeasible analytical models

- Network and communication performance
  - Typical performance indicators (throughput, delay, …)
  - Traffic models and quality of service



- SDN-based monitoring systems
  - Specialized applications running on the SDN controller
  - Anomaly detection and enforcement of mitigation policies
  - Evaluate accuracy, reactiveness and effectiveness

- Cyber/physical security attacks
  - Effects and impact on the network and applications
  - Attack ranking based on effect severity

Our simulation tool

- Goal: design a simulation tool for SDN network
  - Enable quantitative evaluation of performance and security
  - Intended for network designers and researchers

- Built on top of INET/OMNeT++
  - Support for SDN units and OpenFlow
  - Support for evaluation of cyber/physical attacks
  - Work in progress – Source code available at [1]

- This tool does NOT:
  - Discover new attacks and vulnerabilities
  - Evaluate feasibility and success rate of security attacks

[1] https://github.com/marco-tiloca-sics/INET_SDN_dev

INET support for SDN

- Some software modules previously developed [2]
  - Basic SDN Controller and switch nodes
  - Basic OpenFlow messages (exchange and processing)
  - Basic packet-matching with installed flows (based on MAC address only)

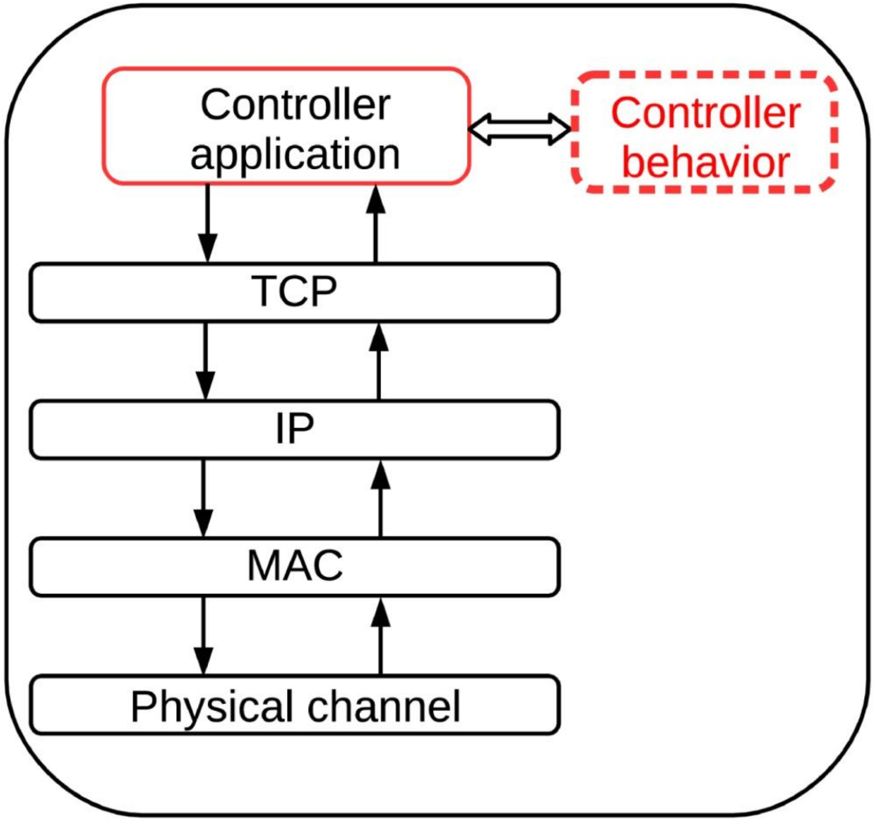
- We have further added
  - OpenFlow messages for flow management and update
  - OpenFlow messages for statistic collection (basic OpenFlow method)
  - Arbitrary complex packet-matching with installed flows

- Future extensions
  - Advanced methods for statistic collection (e.g. *sFlow*)
  - Modules supporting well-known Controller applications

[2] D. Klein and M. Jarschel, *An OpenFlow extension for the OMNeT++ INET framework*", 6th International ICST Conference on Simulation Tools and Techniques (SimuTools '13), pp. 322–329, March 2013

SDN controller

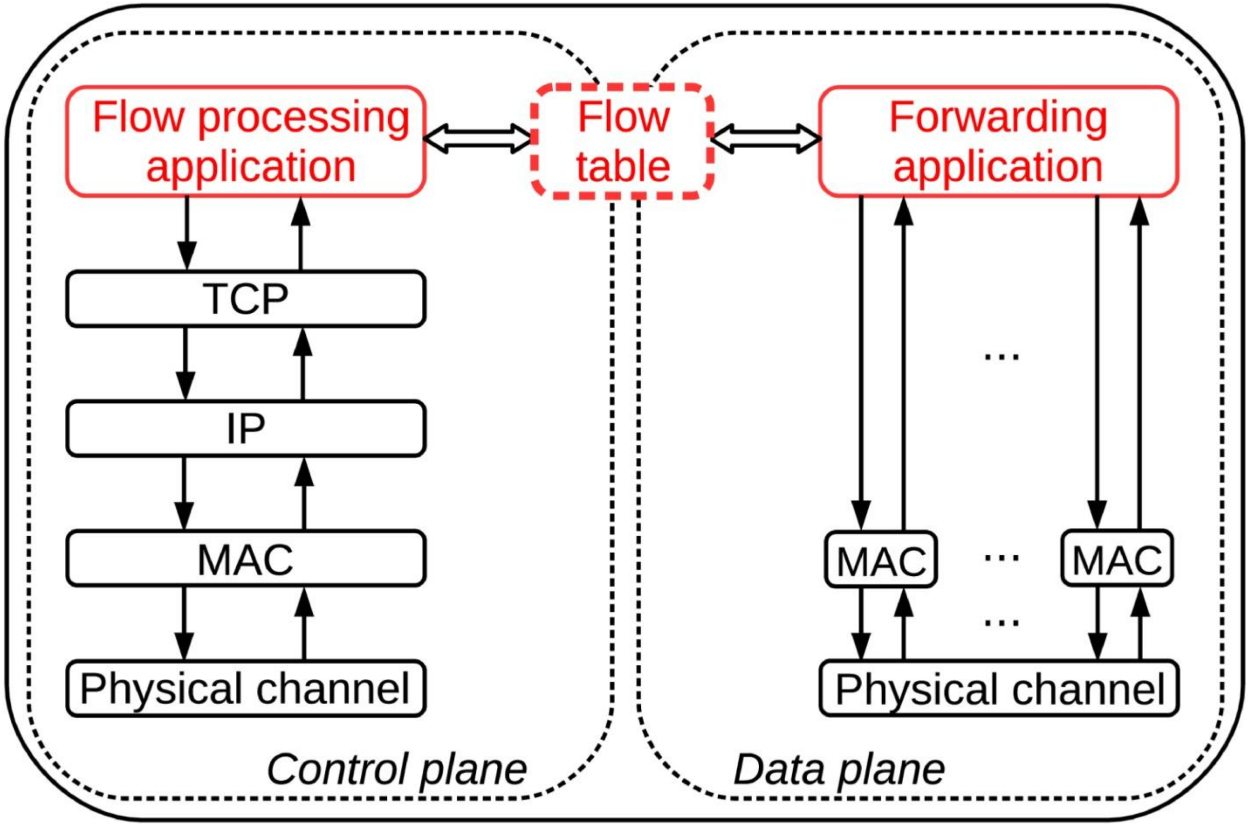
- Host running specific SDN services
- Controller application
  - Flow establishment
  - Installation /update of flows on switches
  - Statistic collection from switches
  - Enforcement of traffic policies
  - …
- Monitoring system
  - Yet another dedicated application
  - Traffic monitoring and anomaly detection
  - Anomaly mitigation and neutralization
  - (more details soon…)



SDN controller node

SDN switch

- Control plane
  - Traditional-host stack
  - Interaction with the SDN controller
- Data plane
  - Collection of minimal stacks
  - Packet matching and forwarding



Switch node

SDN-based monitoring systems

- Step 1 – Statistic collection from switches
  - Basic OpenFlow method based on polling interval
  - Alternative fine-grained methods e.g. sFlow (future work)
- Step 2 – Statistic analysis
  - Dedicated application on the SDN controller
  - Detection of anomalous traffic distribution (e.g. entropy-based)
  - Detection of anomalous traffic volumes to/from network nodes
- Step 3 – Anomaly mitigation
  - Flow installation/update on switches
  - Isolation of anomalous/malevolent traffic

- Basic methods implemented as a proof-of-concept

Evaluation of security attacks

- Attack **effects** are simulated
  - Attacks are assumed to be successfully performed
  - There is no reproduction of their actual execution
  - Only final effects are reproduced at runtime
- Quantitative evaluation
  - Assess effects and impact on networks and applications
  - Observe changes in performance indicators
  - Consider an attack-free case as comparative baseline

- Core concepts first introduced in [3]
  - **Attack Specification Language** and **Attack Simulation Engine**
  - Current adaptation to SDN architectures and scenarios
  - Enable attacks where switches are victims or exploited units

[3] M. Tiloca, F. Racciatti and G. Dini, *Simulative Evaluation of Se-curity Attacks in Networked Critical Infrastructures*, The 2nd International Workshop on Reliability and Security Aspects for Critical Infrastructure Protection (ReSA4CI), Lecture Notes in Computer Science LNCS 9338. Springer, pp. 314–323, September 2015

Core concept #1 - Attack Specification Language

- The user describes attacks to be evaluated
  - Attacks are described in terms of their final effects
  - No need to describe how attacks are actually executed

- Attack format
  - List of atomic events to be injected at runtime
  - Events modeled by high-level *primitive* functions

- Node primitives
  - Intended for physical attacks
  - End targets are network nodes

- Message primitives
  - Intended for cyber attacks
  - End targets are network packets

Core concept #1 - Attack Specification Language

- Physical attacks
  - One node primitive each

Node primitives

destroy() *move()*

- Cyber attacks
  - List of message primitives
  - Packet fields addressed by a *dot* notation
  - Either conditional or unconditional

Message primitives

drop() *create()*
clone() *retrieve()*
change() *send()* *put()*

- Conditional cyber attacks
  - Occur if a condition is verified as true

from *T* ***nodes*** = *<list of nodes>* ***do*** *{*
filter *(<condition>) <list of events>*
}

- Unconditional cyber attacks
  - Occur periodically, from a specified time

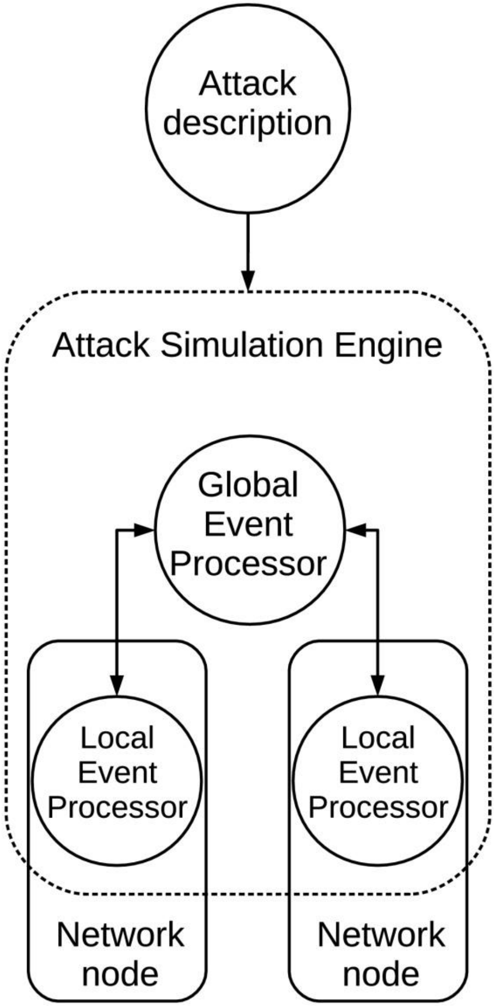
from *T* ***every*** *P* ***do*** *{*
<list of events>
}

Core concept #2 - Attack Simulation Engine

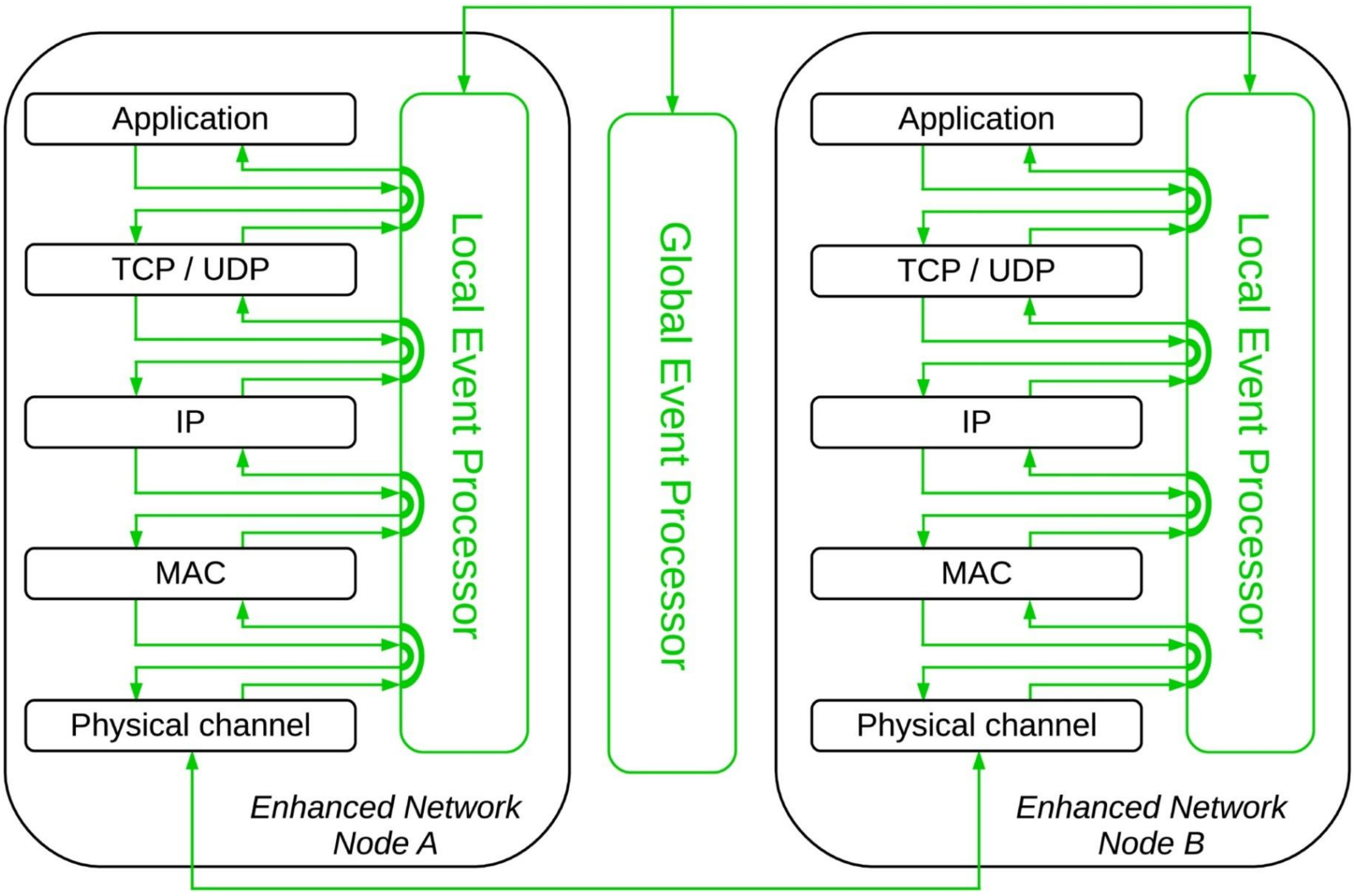
- Additional INET modules
  - Global Event Processor
  - Local Event Processor (1 per network node)
  - Injection and processing of attack events at runtime

- Local Event Processor
  - Gate by-pass between each pair of layers in the stack
  - Intercept, chance and inject packets at different layers
  - Transparent to the network nodes

- Global Event Processor
  - Connected with all the Local Event Processors
  - Enable complex attacks involving more nodes (e.g. wormhole)

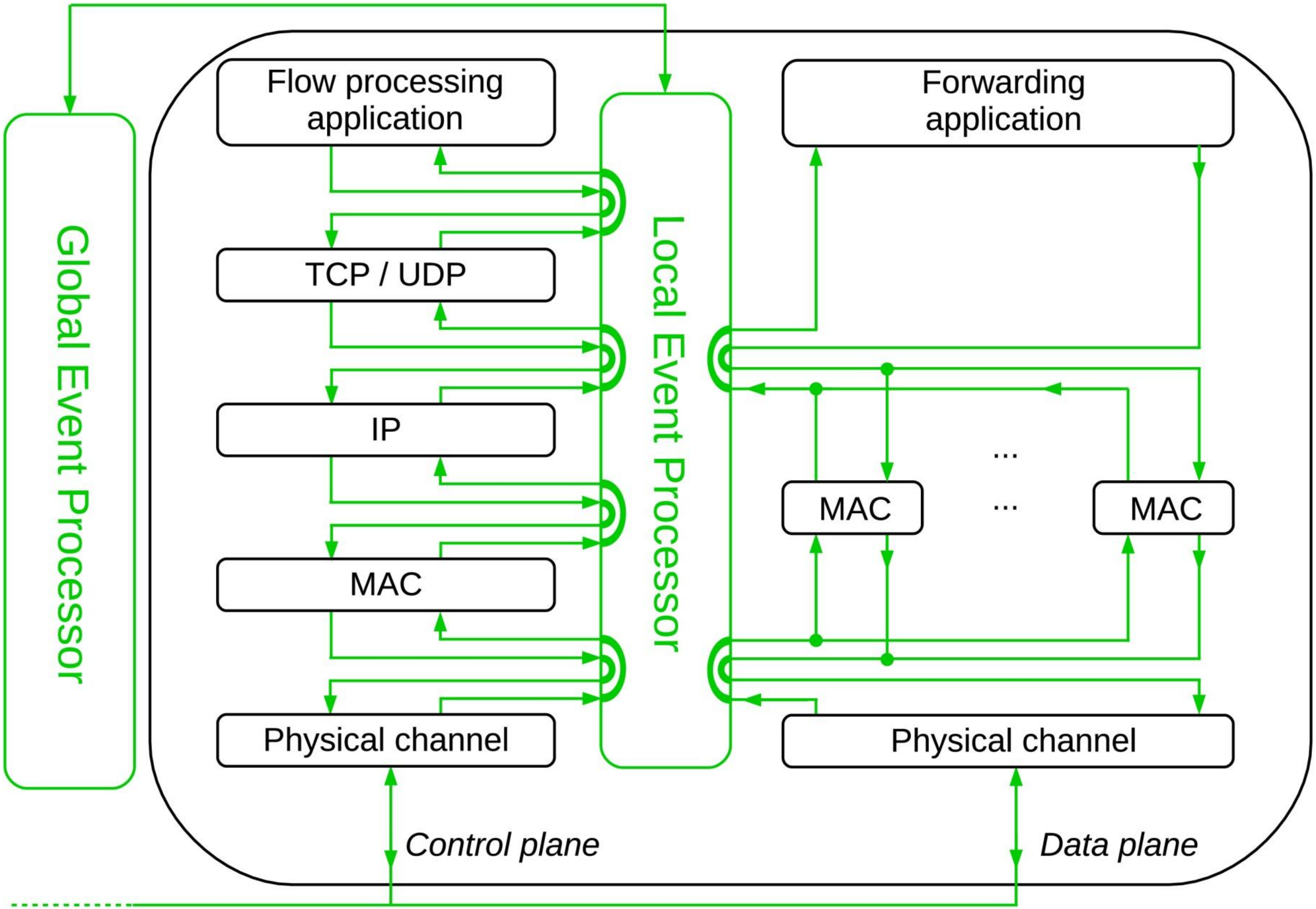


Core concept #2 - Attack Simulation Engine



Adaptation to generic hosts and SDN Controllers

Core concept #2 - Attack Simulation Engine



Adaptation to SDN switches (work in progress)

Reproduction of attack effects

1. The user:
   - Describes the attacks with the specification language
   - Converts the description into XML (Python interpreter)
   - Runs a new simulation importing the XML attack file

2. The Attack Simulation Engine:
   - Parses the XML attack file
   - Builds attack lists and starts attack timers
   - Injects the specified attack events at runtime

3. Collection and analysis of results
   - Attack-free scenario as comparison baseline
   - Attack ranking and selection of countermeasures

USER
Attack description

INET
Injection of attack events

USER
Analysis of results

Reproduction of attack effects

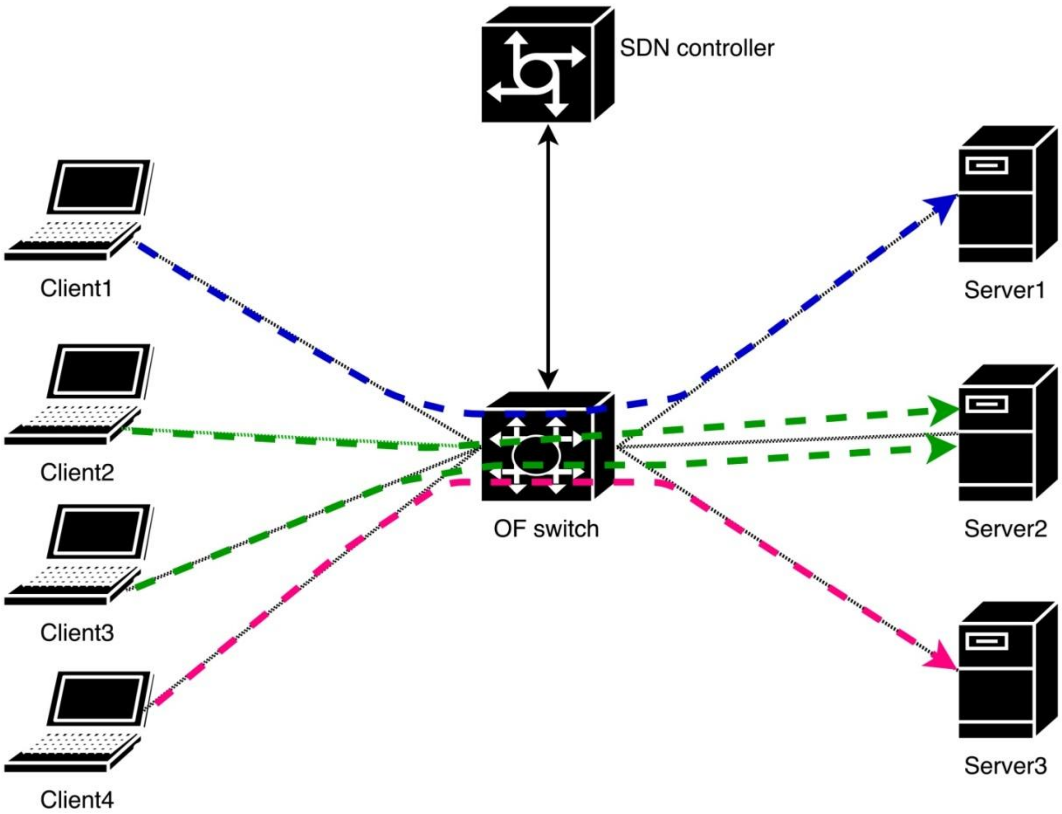
- We have NOT:
  - Modified event scheduling/handling in INET
  - Modified applications or communication protocols

- The user is NOT required to:
  - Implement actual adversaries and attack executions
  - Modify applications and communication protocols
  - Implement or customize INET components

- The user considers as starting points:
  - The network scenario, applications and protocols
  - The applications and service running on the SDN controller
  - The security attacks to be evaluated

Example scenario

- Communication patterns
  - C1→S1 10 pkt/s
  - C2→S2 5 pkt/s
  - C3→S2 3.33 pkt/s
  - C4→S3 5 pkt/s

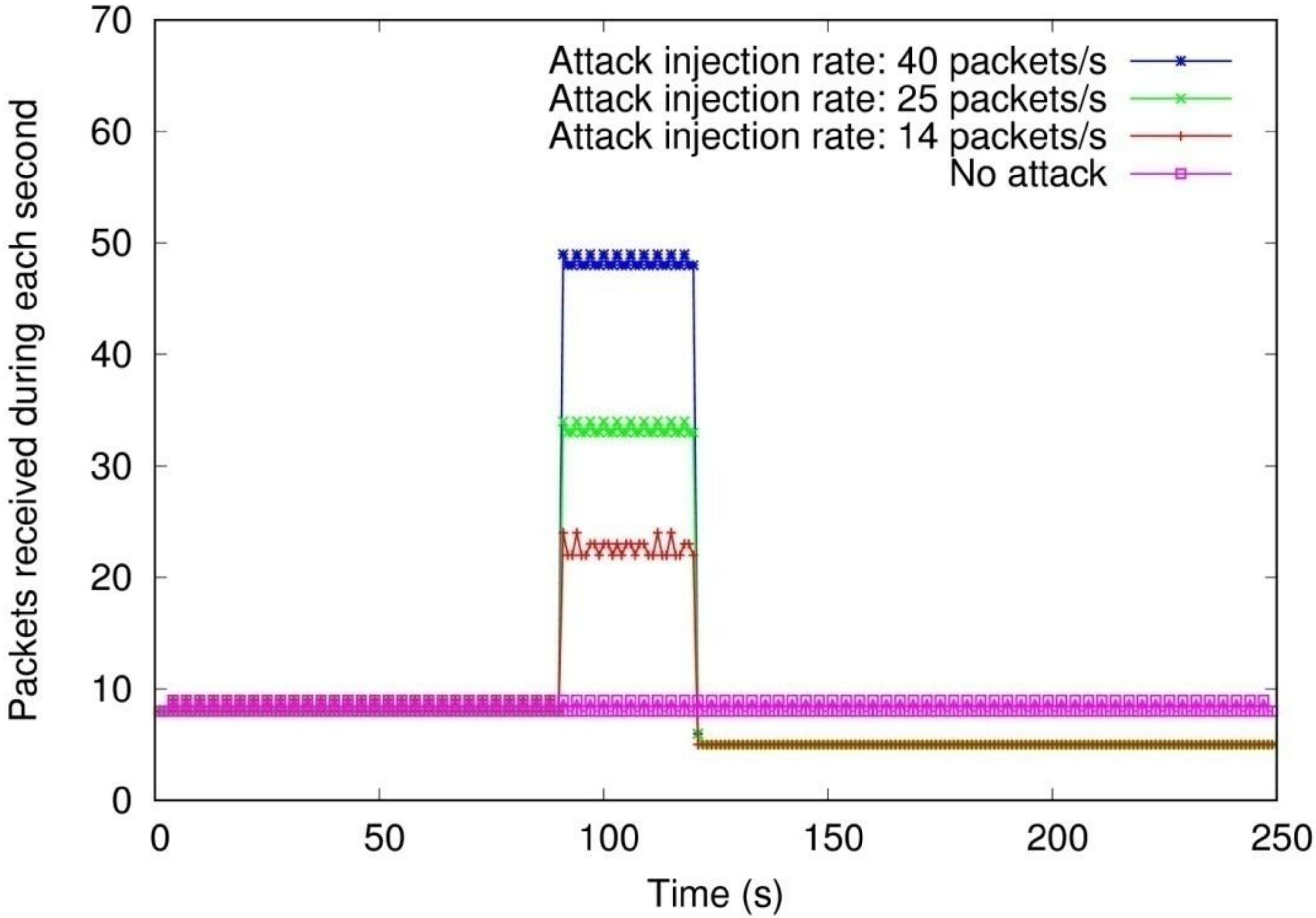


- Flow management policies
  - Periodic expiration (every 30 s)
  - Periodic statistic collection
  - Privacy by design

- Anomaly detection
  - Entropy-based w/ fixed threshold
  - Bounded TX/RX rates per node

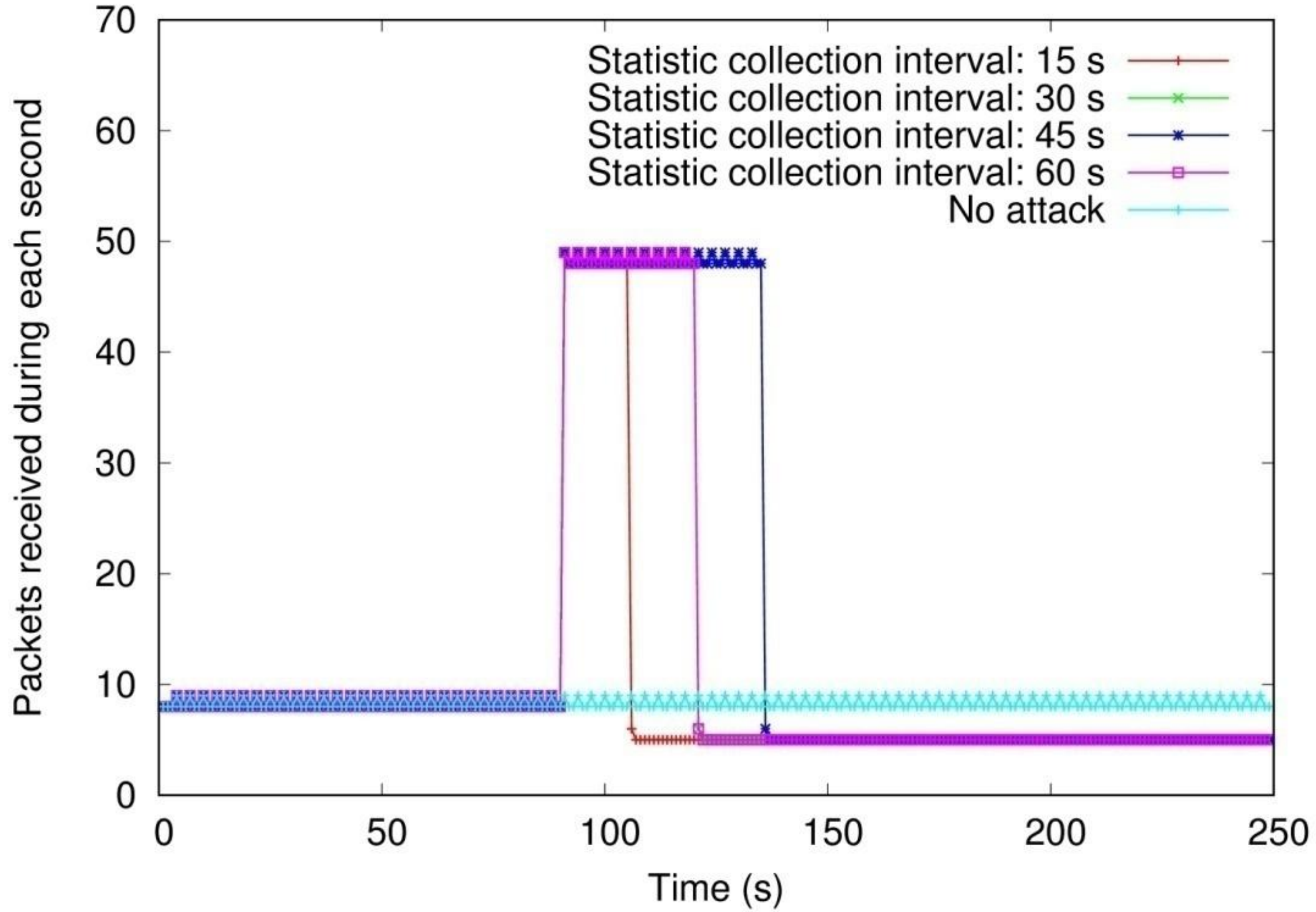
- Denial of Service attack
  - Start at t = 90 s
  - C3 sends additional packets to S2
  - Attack injection rate R

Denial of Service - Results



- Different attack injection rates
  - The stronger the attack, the more packets received by the victim
  - Well-tuned monitoring system: attack always detected at t = 120 s

Denial of Service - Results



- Different interval for statistic collection
  - Well-tuned monitoring system: attack always detected
  - More frequent collections support a faster anomaly detection

Conclusion

- SDN simulation tool based on INET
  - Evaluation of typical performance indicators
  - Evaluation of SDN-based monitoring systems
  - Evaluation of impact and effects of security attacks

- Attack evaluation
  - Attack described by a high-level specification language
  - Sequence of atomic events injected at runtime
  - No need to implement actual adversaries or attack execution

- Future works
  - Support for additional attacks (switches as victims or attack vectors)
  - Evaluation of different classes of security attacks
  - Support advanced methods for statistic collection
  - Support well-known applications for SDN controllers

Thanks for your attention!

Marco Tiloca
marco@sics.se
http://www.sics.se/~marco