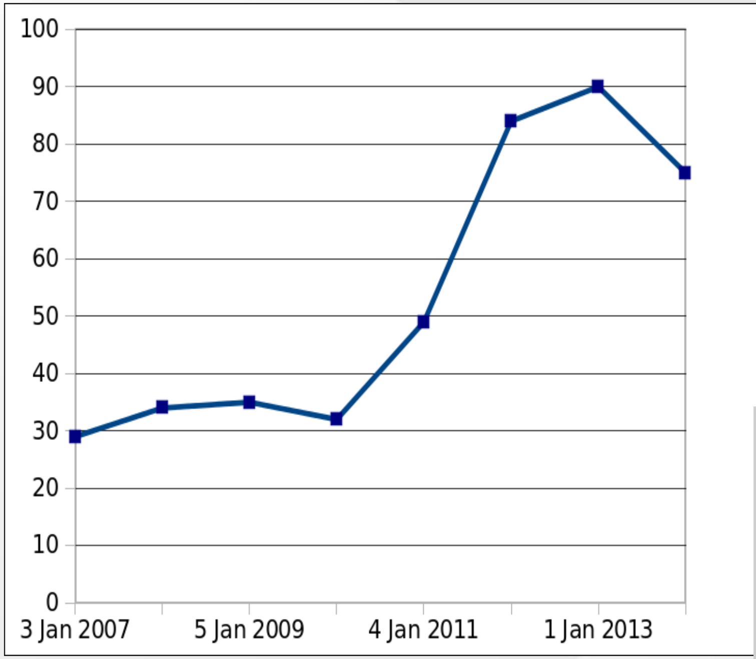
ARP Mitigation at AMS-IX

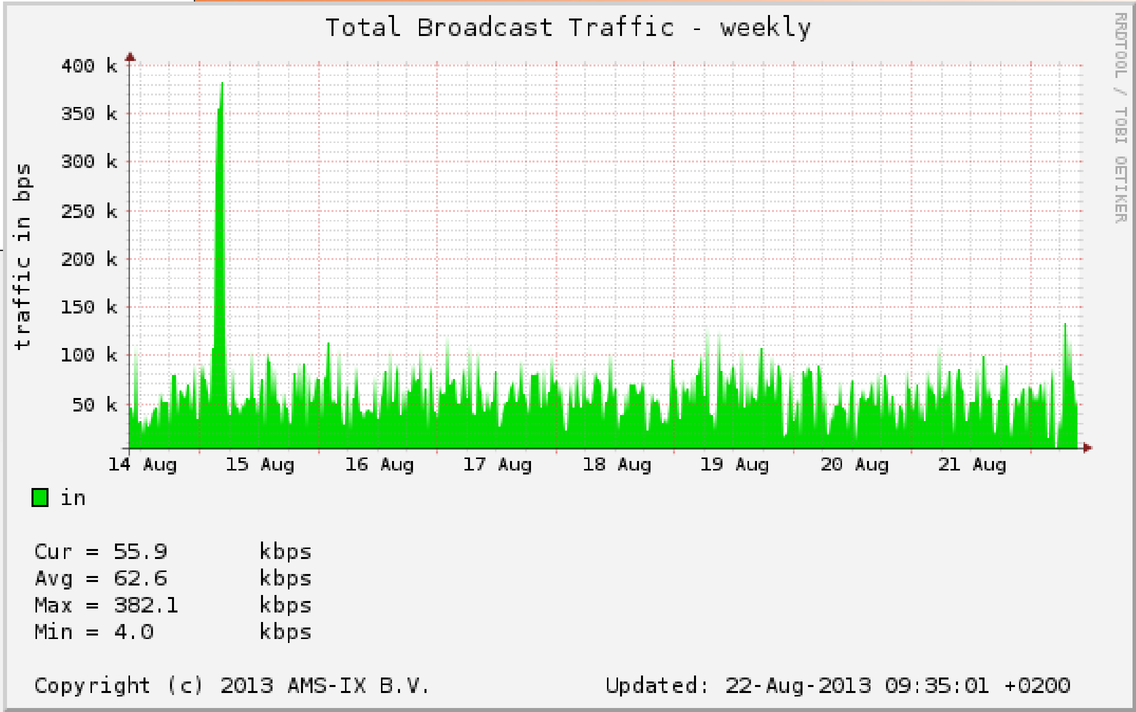
A practical use case for OpenFlow

Martin Pels
martin.pels@ams-ix.net

03/10/2013

The AMS-IX Peering LAN

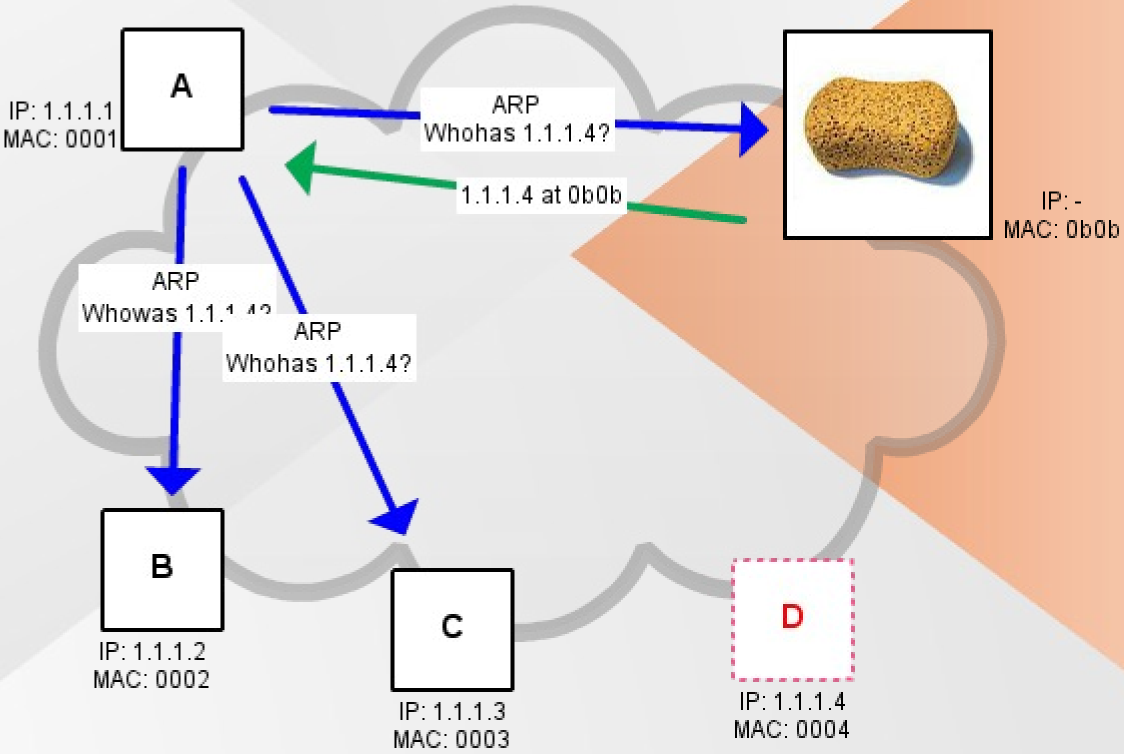




```
Cur = 55.9       kbps
Avg = 62.6       kbps
Max = 382.1      kbps
Min = 4.0        kbps

Copyright (c) 2013 AMS-IX B.V.            Updated: 22-Aug-2013 09:35:01 +0200
```

ARP sponge


A
IP: 1.1.1.1
MAC: 0001
ARP
Whohas 1.1.1.4?
1.1.1.4 at 0b0b
IP: -
MAC: 0b0b
ARP
Whowas 1.1.1.4?
ARP
Whohas 1.1.1.4?
B
IP: 1.1.1.2
MAC: 0002
C
IP: 1.1.1.3
MAC: 0003
D
IP: 1.1.1.4
MAC: 0004

Challenges

- CPU use in customer routers
- ARP cache/next-hop table limitations
- CPU use in AMS-IX PEs
- ARP spoofing (software bugs)

SDN to the rescue!

Source: OpenDaylight

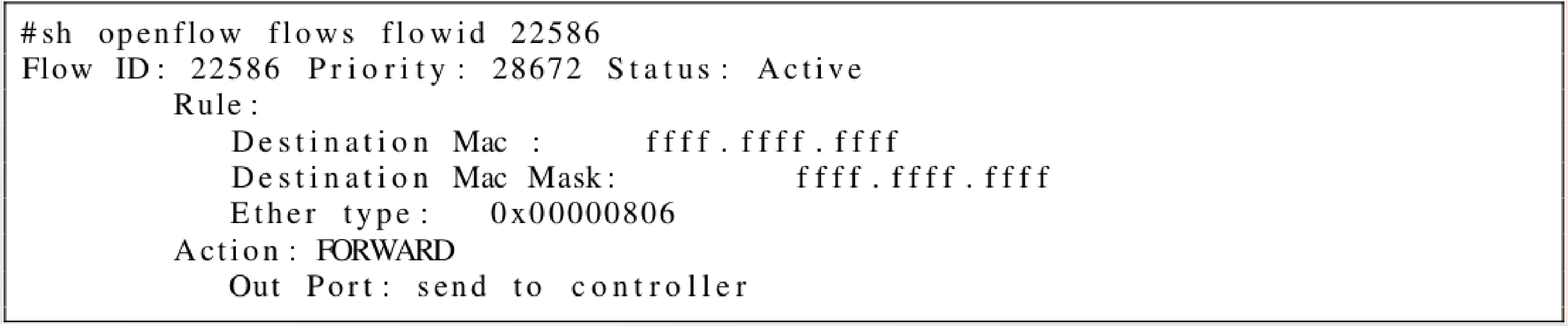
Requirements

- Integrate into existing MPLS/VPLS environment
- Scalability
- Stability
  - No single point of failure
  - No potential impact on control functions

Research based on OpenFlow v1.0.0 / Brocade NetIronXMR-MLX 05500b

OpenFlow-hybrid

- Route traffic to either OpenFlow pipeline or normal pipeline
- Classification mechanism not standardized
- Brocade: Hybrid Port Mode
  - Flow match? → Execute flow actions
  - No match → Submit to normal pipeline



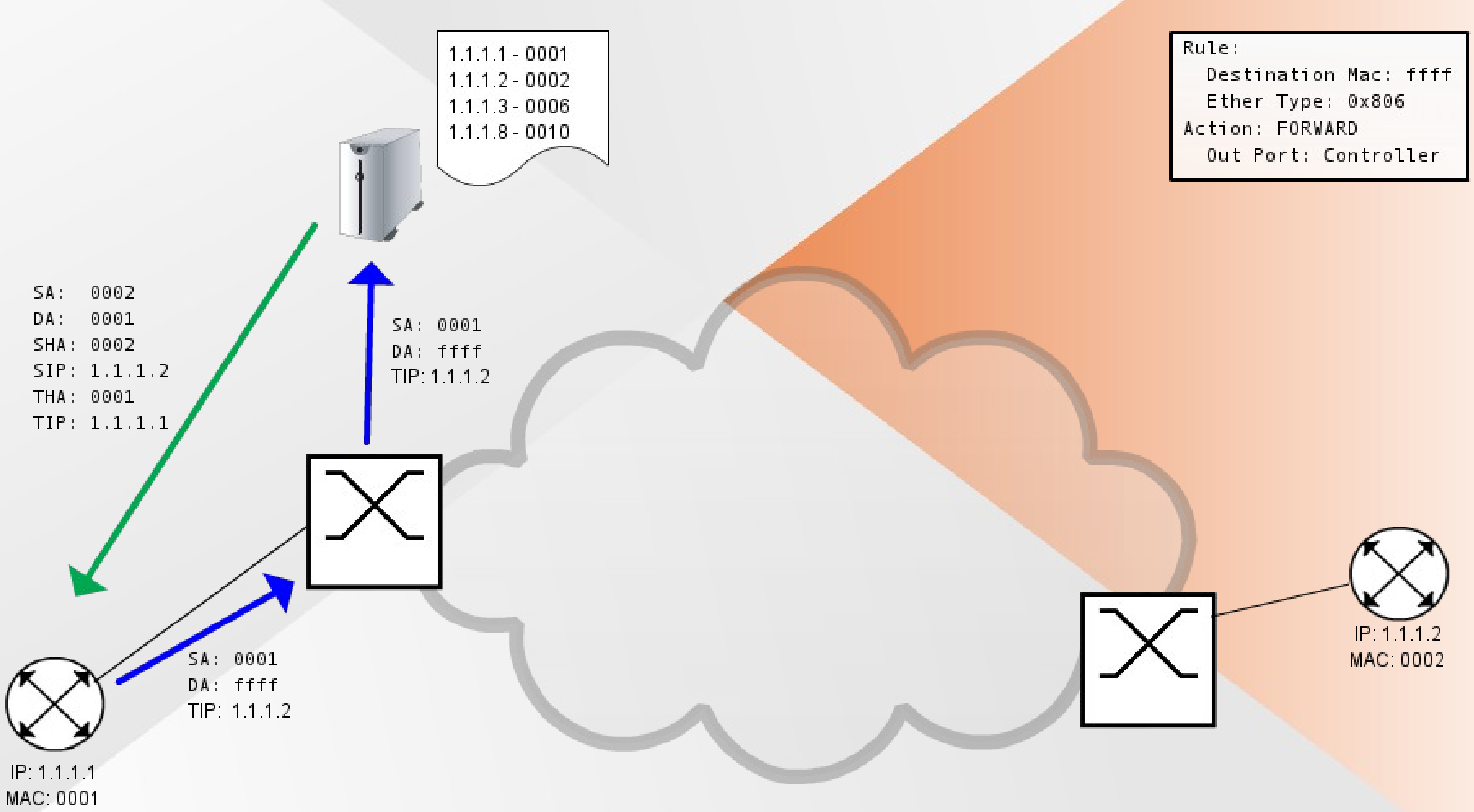
```
#sh openflow flows flowid 22586
Flow ID: 22586 Priority: 28672 Status: Active
        Rule:
           Destination Mac :      ffff.ffff.ffff
           Destination Mac Mask:          ffff.ffff.ffff
           Ether type:   0x00000806
        Action: FORWARD
           Out Port: send to controller
```

Solutions

1. Controller answers from static ARP table
2. Controller answers on behalf of client (dynamic ARP table)
3. Customer router answers (controller forwards request as unicast)
4. ARP Sponge answers (flowrule forwards request to sponge)
5. Customer router answers (flowrule forwards request as unicast)

Controller processing

(Static ARP table)



```
1.1.1.1 - 0001
1.1.1.2 - 0002
1.1.1.3 - 0006
1.1.1.8 - 0010
```

```
Rule:
  Destination Mac: ffff
  Ether Type: 0x806
Action: FORWARD
  Out Port: Controller
```

```
SA:  0002
DA:  0001
SHA: 0002
SIP: 1.1.1.2
THA: 0001
TIP: 1.1.1.1
```

```
SA: 0001
DA: ffff
TIP: 1.1.1.2
```

```
SA: 0001
DA: ffff
TIP: 1.1.1.2
```

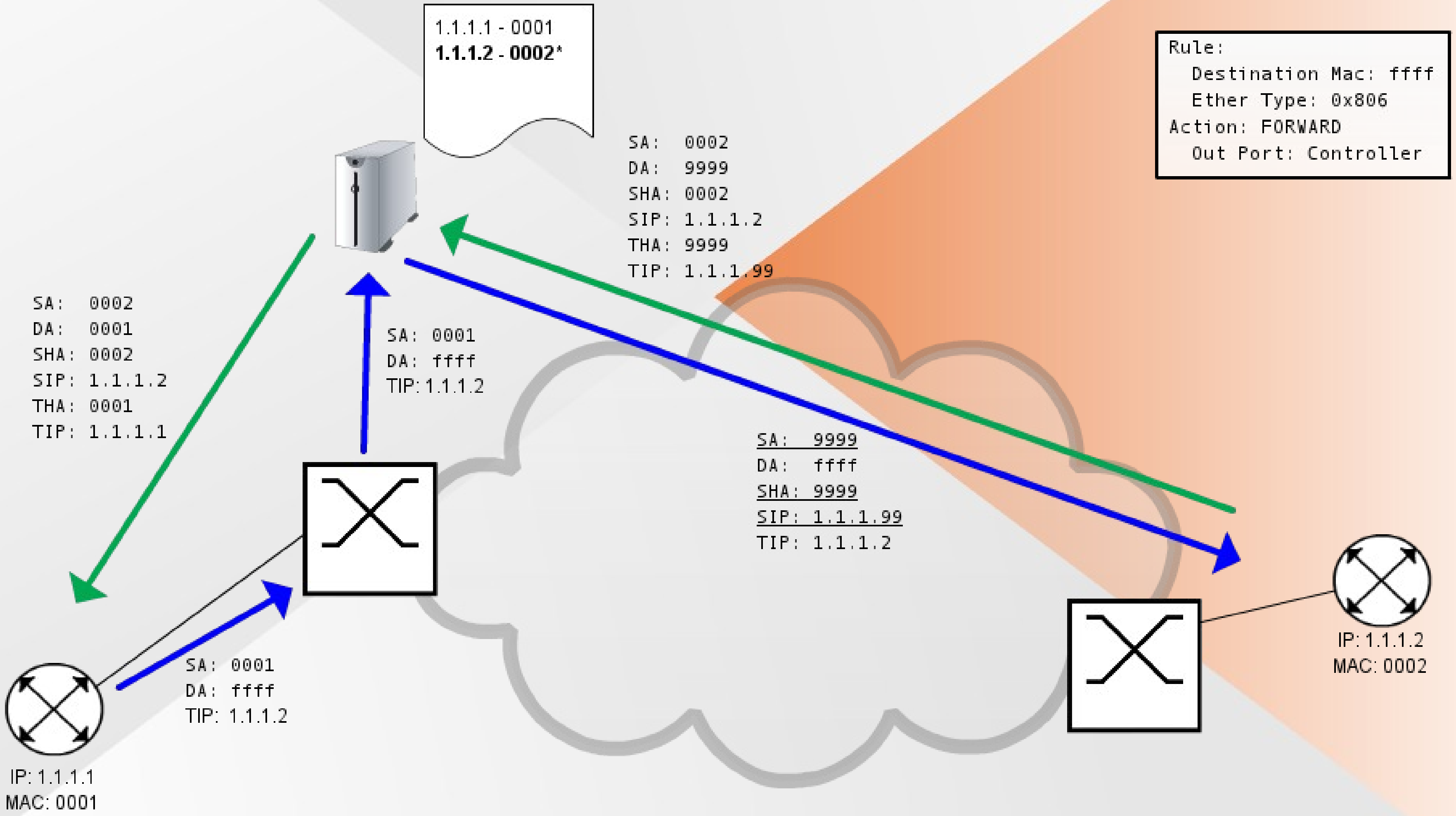
IP: 1.1.1.1
MAC: 0001

IP: 1.1.1.2
MAC: 0002

Controller processing

(Dynamic ARP table)

```
Rule:
  Destination Mac: ffff
  Ether Type: 0x806
Action: FORWARD
  Out Port: Controller
```



1.1.1.1 - 0001
1.1.1.2 - 0002*

```
SA:  0002
DA:  9999
SHA: 0002
SIP: 1.1.1.2
THA: 9999
TIP: 1.1.1.99
```

```
SA:  0002
DA:  0001
SHA: 0002
SIP: 1.1.1.2
THA: 0001
TIP: 1.1.1.1
```

```
SA: 0001
DA: ffff
TIP: 1.1.1.2
```

```
SA:  9999
DA:  ffff
SHA: 9999
SIP: 1.1.1.99
TIP: 1.1.1.2
```

IP: 1.1.1.2
MAC: 0002

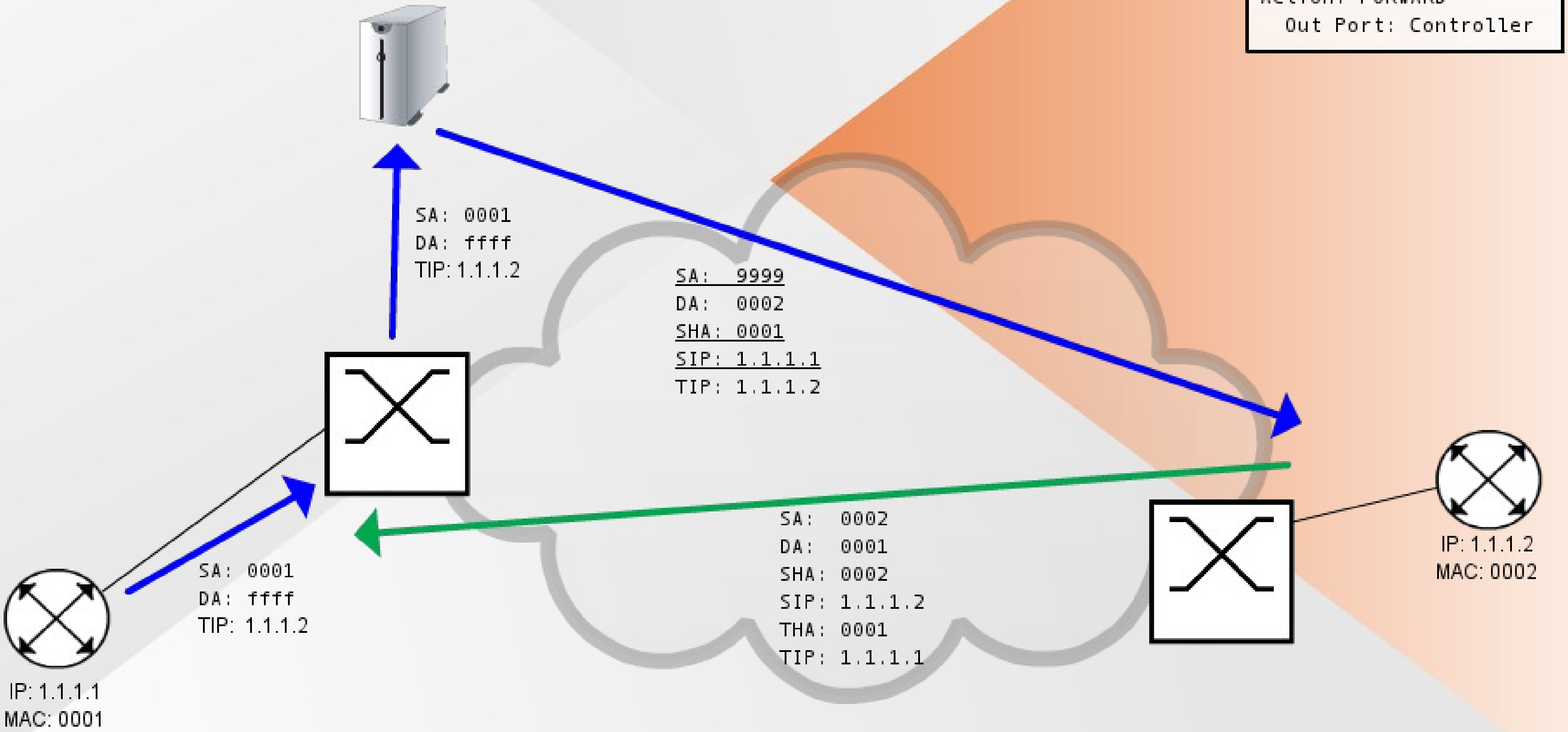
```
SA: 0001
DA: ffff
TIP: 1.1.1.2
```

IP: 1.1.1.1
MAC: 0001

Controller processing

(Forward to customer)

```
Rule:
  Destination Mac: ffff
  Ether Type: 0x806
Action: FORWARD
  Out Port: Controller
```



SA: 0001
DA: ffff
TIP: 1.1.1.2

SA: 9999
DA: 0002
SHA: 0001
SIP: 1.1.1.1
TIP: 1.1.1.2

SA: 0001
DA: ffff
TIP: 1.1.1.2

SA: 0002
DA: 0001
SHA: 0002
SIP: 1.1.1.2
THA: 0001
TIP: 1.1.1.1

IP: 1.1.1.1
MAC: 0001

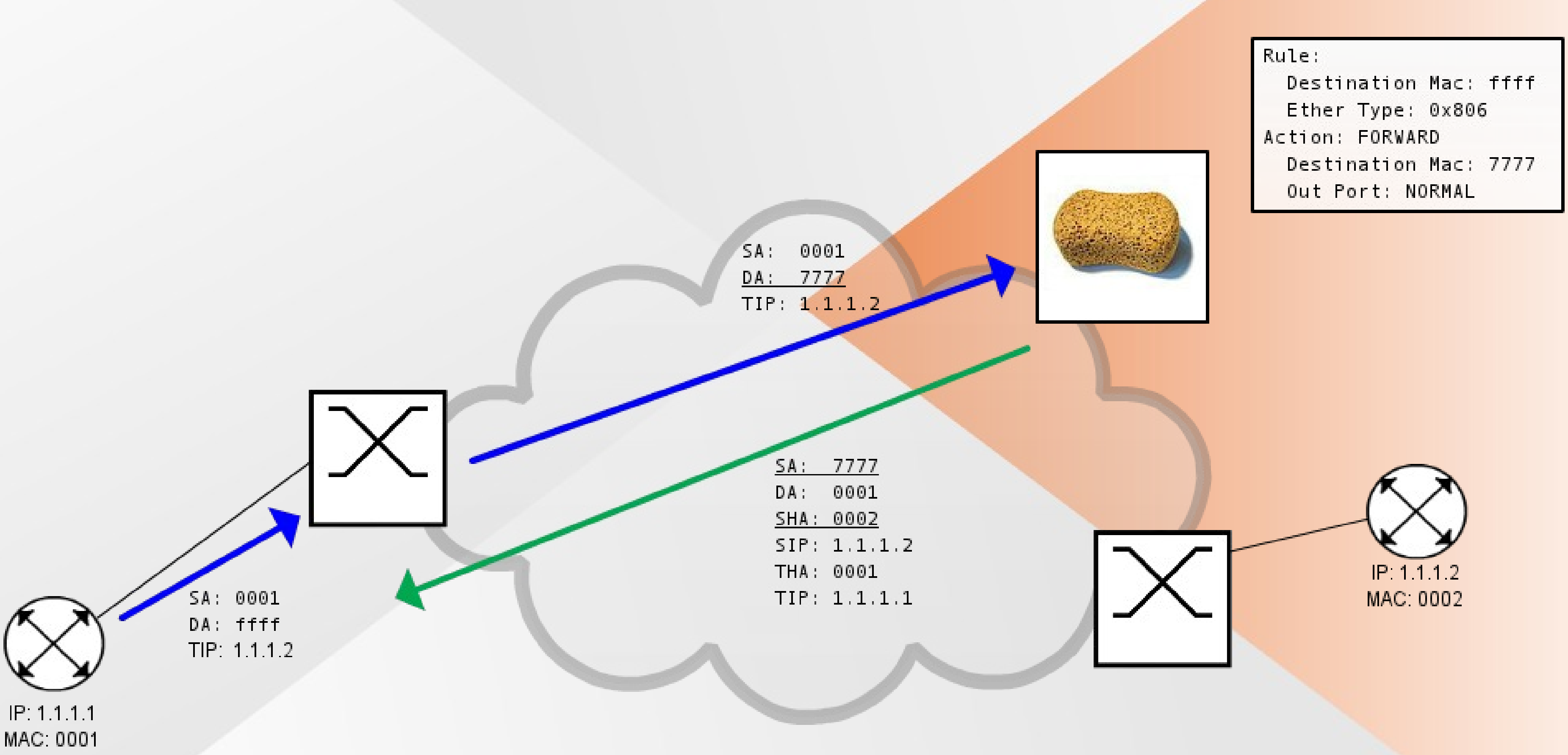
IP: 1.1.1.2
MAC: 0002

Controller processing – observations

- Removes all broadcast traffic
- Implementable today (built Proof of Concept)

- Requires always active controller
- Controller traffic is CPU forwarded
- No rate-limiting of controller traffic available

Send to ARP sponge



```
Rule:
  Destination Mac: ffff
  Ether Type: 0x806
Action: FORWARD
  Destination Mac: 7777
  Out Port: NORMAL
```

SA: 0001
DA: 7777
TIP: 1.1.1.2

SA: 7777
DA: 0001
SHA: 0002
SIP: 1.1.1.2
THA: 0001
TIP: 1.1.1.1

SA: 0001
DA: ffff
TIP: 1.1.1.2

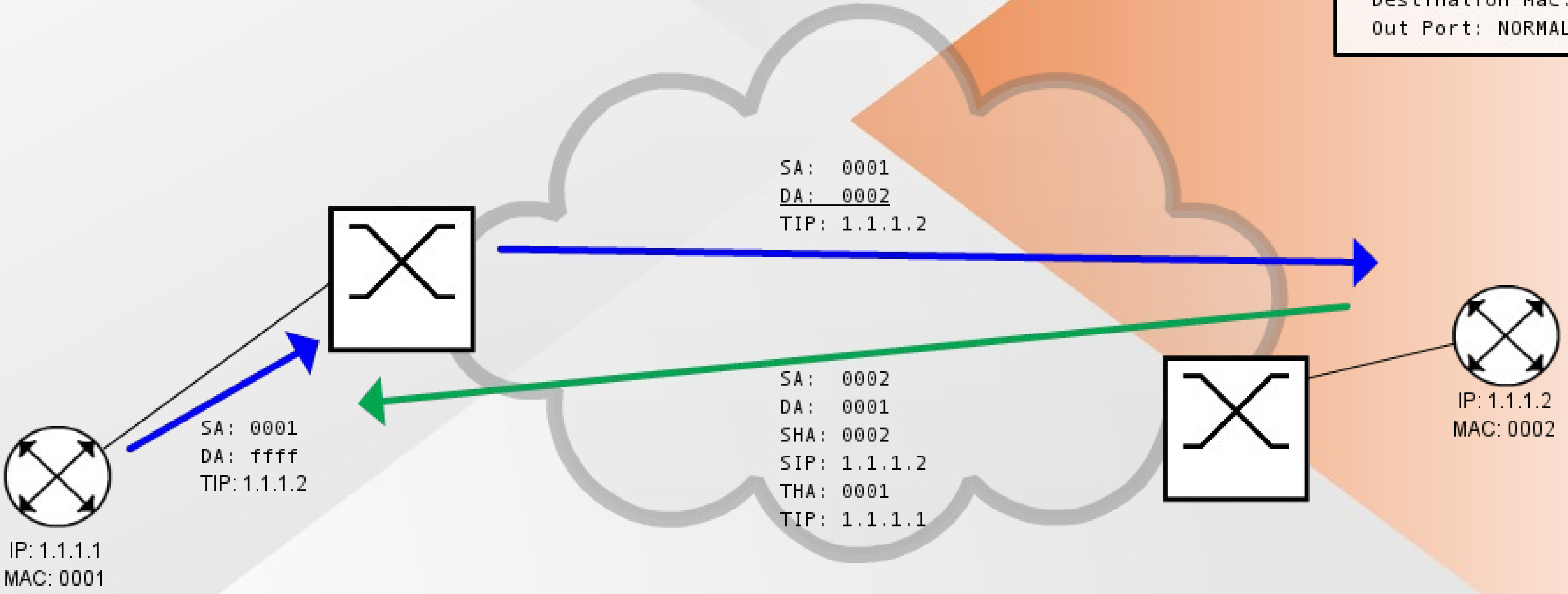
Send to ARP sponge – observations

- Forwarding via flowrule; no dependency on controller
- No CPU forwarding on PE

- Requires always active ARP sponge
- Requires routers to ignore source address in Ethernet header
- Requires support for NORMAL output port (OpenFlow 1.0, optional)
- Would be nice to have metering (Openflow 1.3)

Convert to unicast

```
Rule:
  Destination Mac: ffff
  Ether Type: 0x806
  ARP Target IP: 1.1.1.2
Action: FORWARD
  Destination Mac: 0002
  Out Port: NORMAL
```



SA: 0001
DA: 0002
TIP: 1.1.1.2

SA: 0001
DA: ffff
TIP: 1.1.1.2

SA: 0002
DA: 0001
SHA: 0002
SIP: 1.1.1.2
THA: 0001
TIP: 1.1.1.1

IP: 1.1.1.1
MAC: 0001

IP: 1.1.1.2
MAC: 0002

Convert to unicast – observations (1)

- Forwarding via flowrule
- No dependency on controller or ARP sponge
- Could be extended to block invalid ARP replies
  - Prevention of ARP spoofing

Convert to unicast – observations (2)

- Requests for down routers are still flooded
  - Unknown unicast instead of broadcast (not sent to customer router CPU)
  - Could disable CAM aging to mitigate this
- Broadcast still required for migration scenario's

- Requires support for NORMAL output port (OpenFlow 1.0, optional)
- Requires matching on L2 header + ARP payload (OpenFlow 1.0, optional)
- Would be nice to have metering (OpenFlow 1.3)

Conclusions / Future work

- Solutions exist, but mostly in theory
- Vendor implementation too limited for production use (but we expected this)
- OpenFlow offers a lot, but most features are optional to implement

- Future work
  - IPv6 Neighbor solicitations
  - L2 Flowrules for all traffic?
- Also looking into other alternatives (e.g. E-VPN)

Recommendations

- Operators
  - Limit controller traffic
  - Carefully consider controller placement
  - Investigate now; tell your vendor what you need!
- Vendors
  - Design for in-line use (hybrid mode, NORMAL virtual output port)
  - Design for flexibility (implement **all** the fields)
  - Protect control functions (rate-limit controller traffic or forward in hardware)

Questions?

University of Amsterdam student paper:
http://staff.science.uva.nl/~delaat/rp/2012-2013/p57/report.pdf