Interpreted Active Packets for Ephemeral State Processing Routers

S. Martin G. Leduc

Research Unit in Networking, University of Liège

7th International Working Conference on Active and Programmable Networks

Outline

You Said "Ephemeral State Processing" ?

cf. *Lightweight Network Support for Scalable End-to-End Services*, Kenneth L. Calvert et al., SIGCOMM 2002

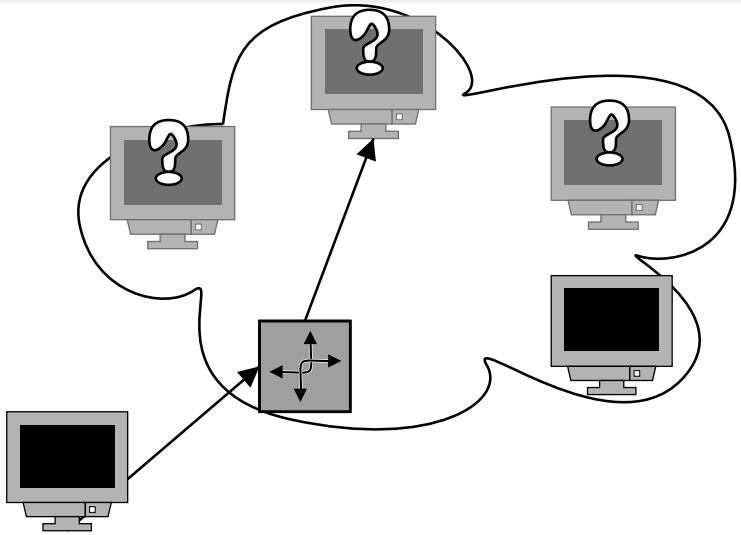
A minimalist active service based on *ephemeral state stores*:

- memory slots on routers with key-based access
- values stored for 10 sec., no refresh.
- ESP defines *operations* on those slots.
- tells if packet is forwarded or dropped.

⇒ Applications in topology discovery, flow aggregation, multicast...

⇒ Can fit network processors such as IXP1200 or IXP2400

Ephemeral State Could Store Much More !



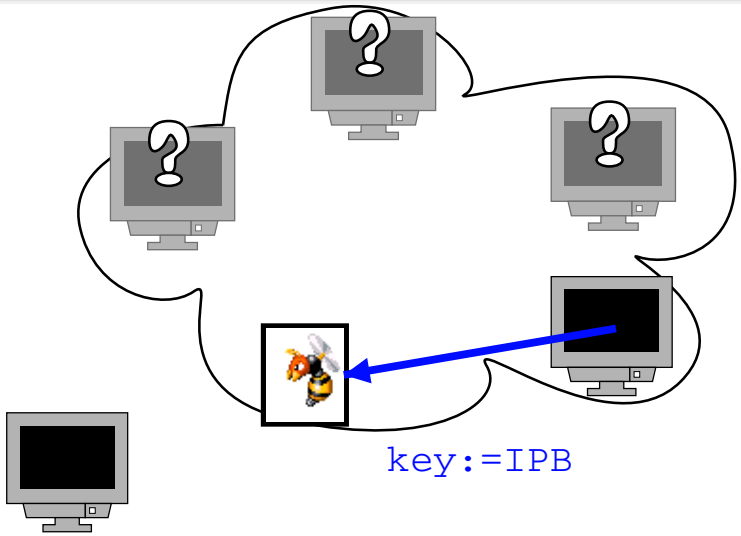
Potential Applications

1. locate “volatile” peers
2. MPEG smart dropping
3. track mobile hosts

Missing Features

- store IP addresses in tags
- early packet return
- interface state inspection
- packet rerouting from tags

Ephemeral State Could Store Much More !



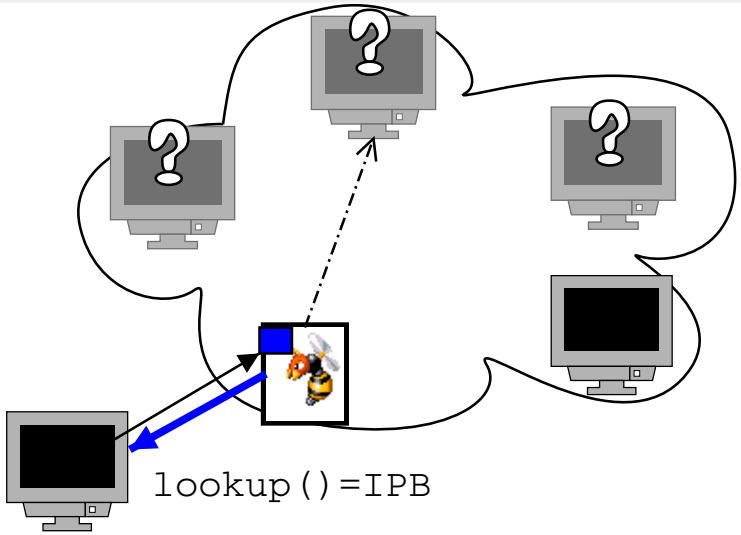
Potential Applications

1. locate “volatile” peers
2. MPEG smart dropping
3. track mobile hosts

Missing Features

- store IP addresses in tags
- early packet return
- interface state inspection
- packet rerouting from tags

Ephemeral State Could Store Much More !



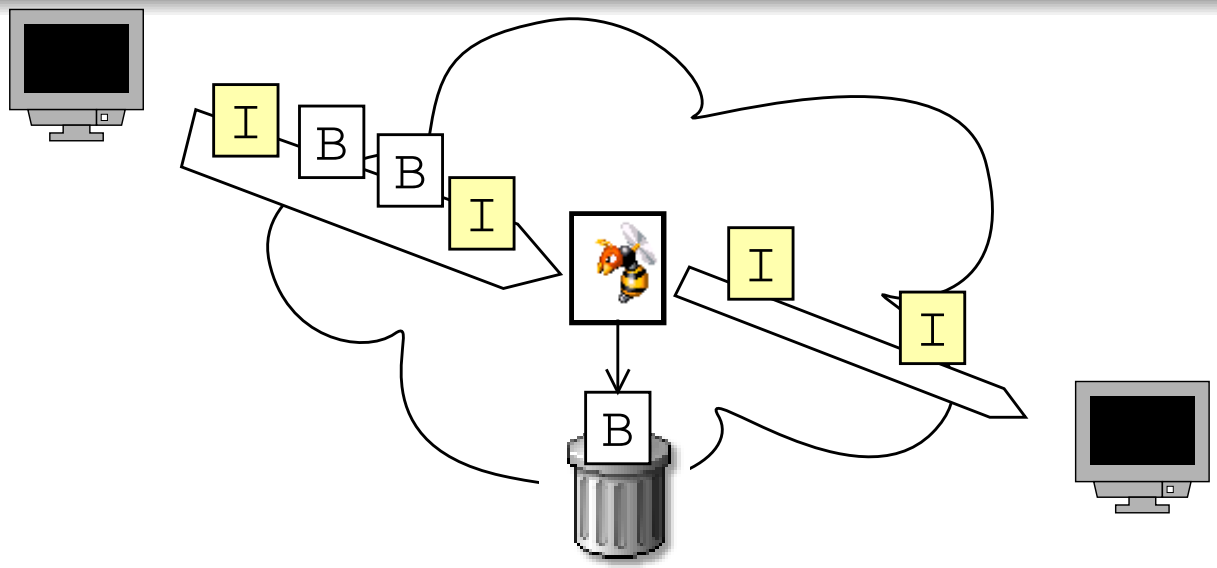
Potential Applications

1. locate “volatile” peers
2. MPEG smart dropping
3. track mobile hosts

Missing Features

- store IP addresses in tags
- early packet return
- interface state inspection
- packet rerouting from tags

Ephemeral State Could Store Much More !



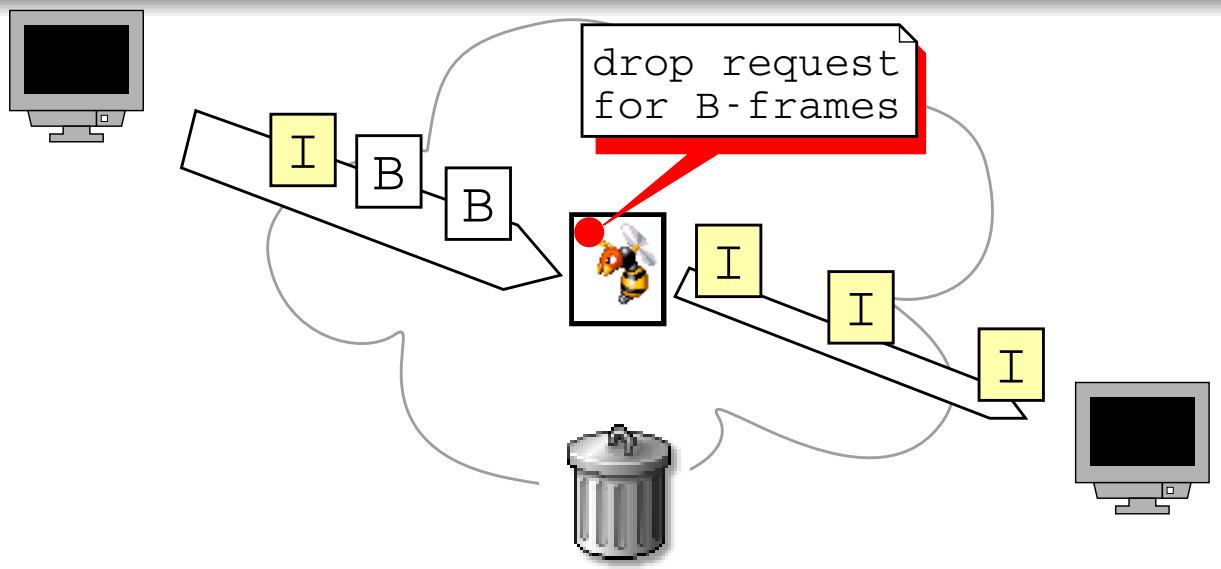
Potential Applications

1. locate “volatile” peers
2. MPEG smart dropping
3. track mobile hosts

Missing Features

- store IP addresses in tags
- early packet return
- interface state inspection
- packet rerouting from tags

Ephemeral State Could Store Much More !



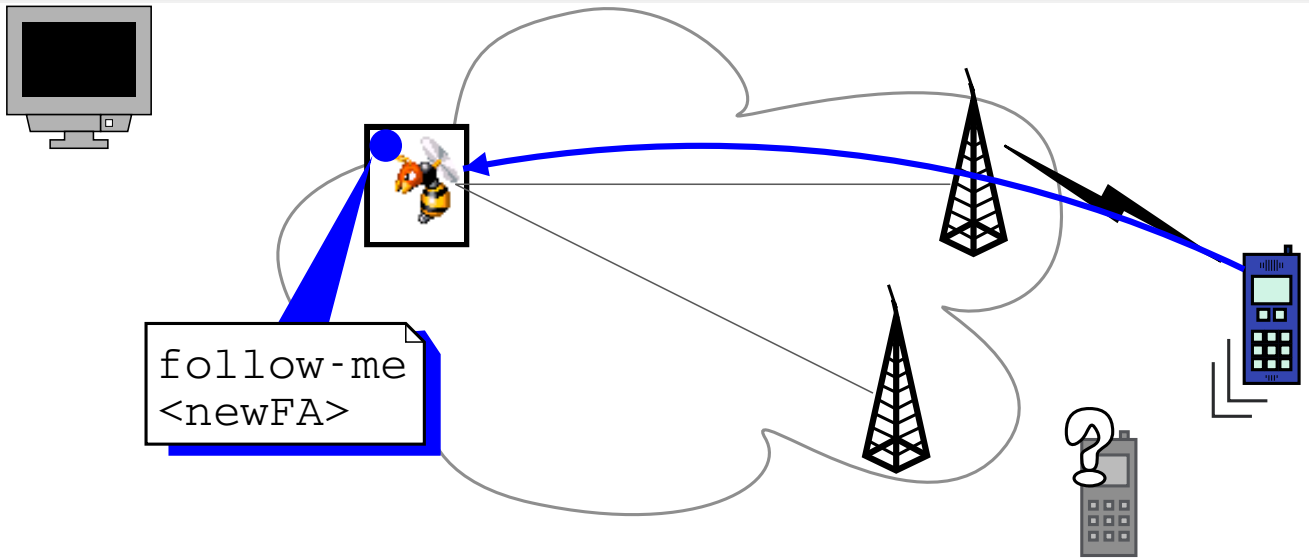
Potential Applications

1. locate “volatile” peers
2. MPEG smart dropping
3. track mobile hosts

Missing Features

- store IP addresses in tags
- early packet return
- interface state inspection
- packet rerouting from tags

Ephemeral State Could Store Much More !



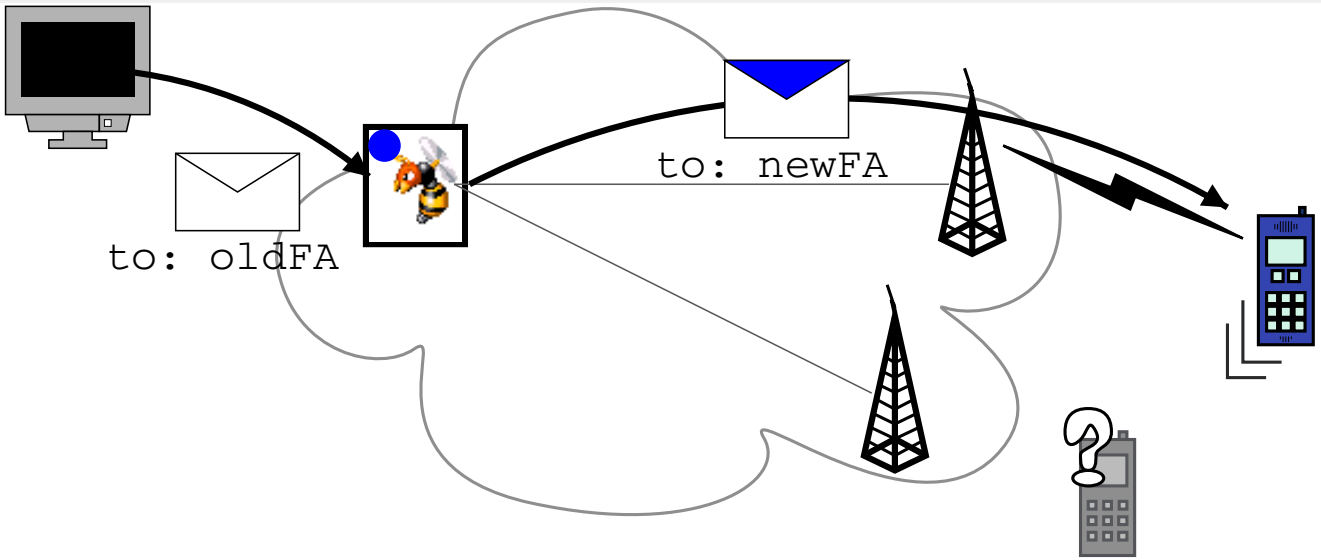
Potential Applications

1. locate “volatile” peers
2. MPEG smart dropping
3. track mobile hosts

Missing Features

- store IP addresses in tags
- early packet return
- interface state inspection
- packet rerouting from tags

Ephemeral State Could Store Much More !



Potential Applications

1. locate “volatile” peers
2. MPEG smart dropping
3. track mobile hosts

Missing Features

- store IP addresses in tags
- early packet return
- interface state inspection
- packet rerouting from tags

Extending ESP Instruction Set

Beyond 'missing features', ESP only support very few operations ...

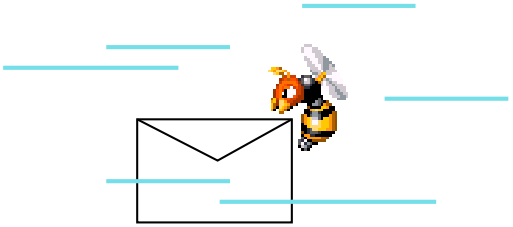
- little space for more ESP code on IXP microengine
- unconvenient to reprogram on the fly (no JIT)

Can we manage to interprete bytecode carried in packets ...

- with comparable packet processing time ?
- without putting the router at risk ?
- and store the interpreter on a microengine ?

World-Friendly Active Packets for ESP

Safe and efficient framework, for both end-user and network operators



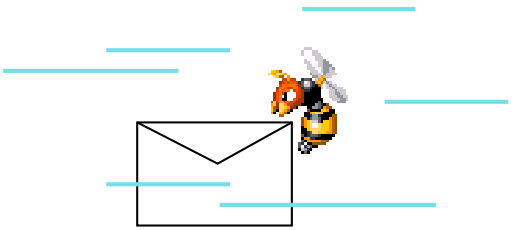
Router-Friendly do not waste resources

Network-Friendly behave like IP packets

User-Friendly don’t perform unexpected actions

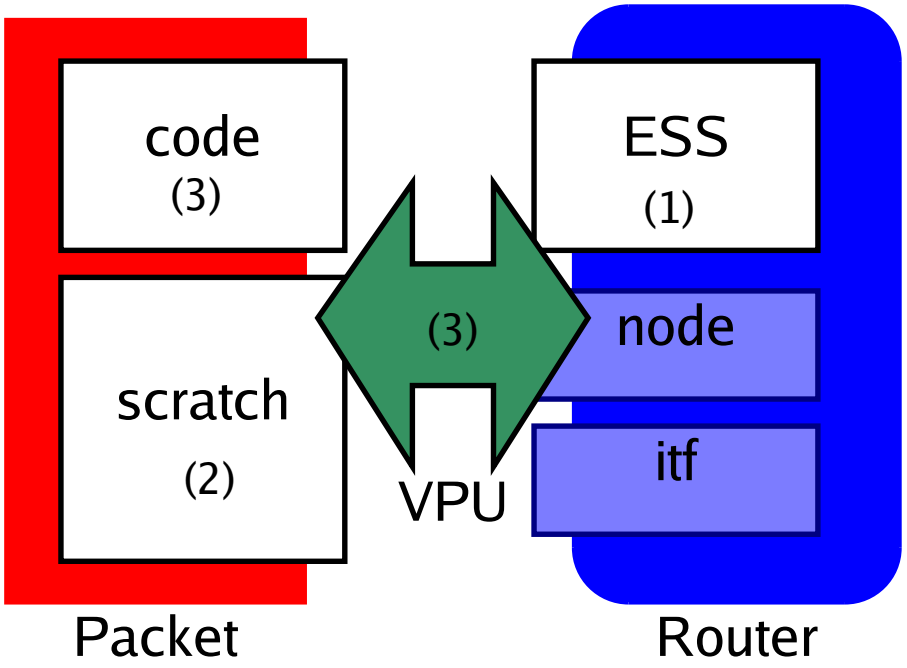
World-Friendly Active Packets for ESP

Limited, yet useful programmability



- small programmable control protocols
- programmable packet control embedded in datapackets
- discover network topology, user communities, services

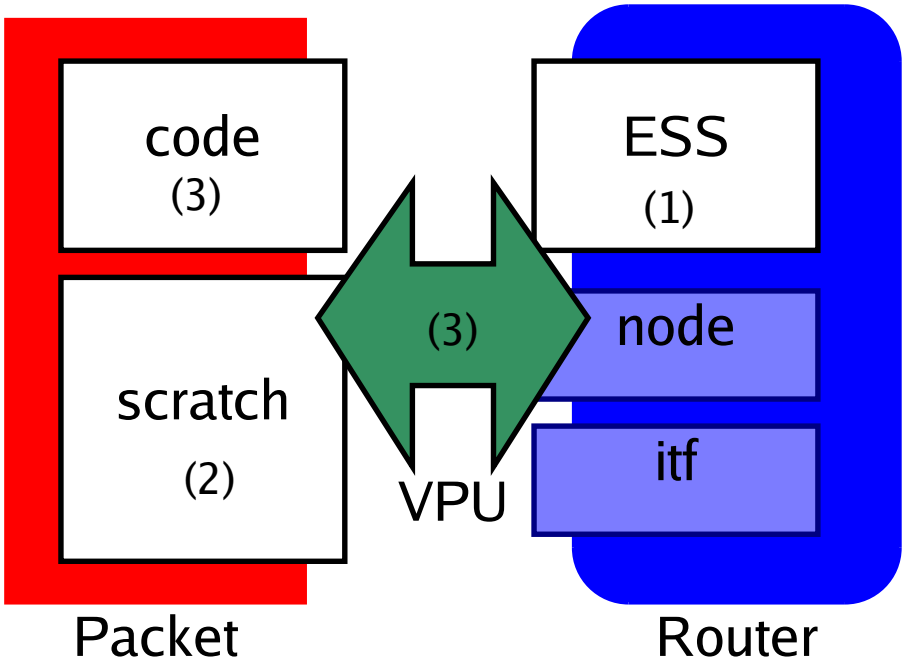
Inside WASP: The Virtual Processing Unit



1. lookup/insert into **ESS**
2. load/store data from packet's scratchpad
3. interpreted, RISC-inspired packet bytecode
4. control opcodes (drop, forward, return)

- tiny interpreter code (4K)
- data and code used 'as is' (no marshalling)
- simple ALU design and compact opcodes

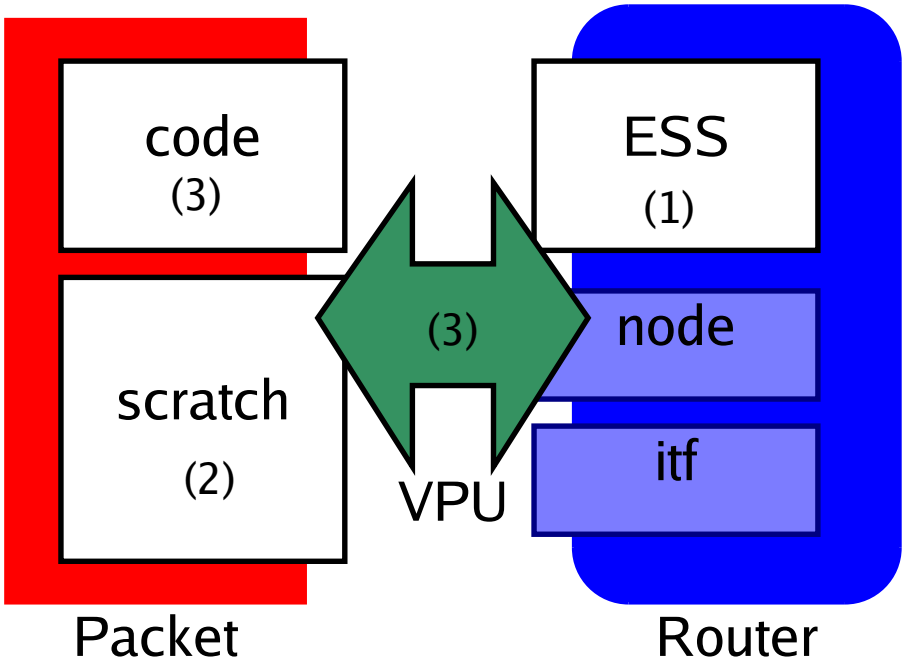
Inside WASP: The Virtual Processing Unit



1. lookup/insert into ESS
2. load/store data from **packet's scratchpad**
3. interpreted, RISC-inspired packet bytecode
4. control opcodes (drop, forward, return)

- tiny interpreter code (4K)
- data and code used 'as is' (no marshalling)
- simple ALU design and compact opcodes

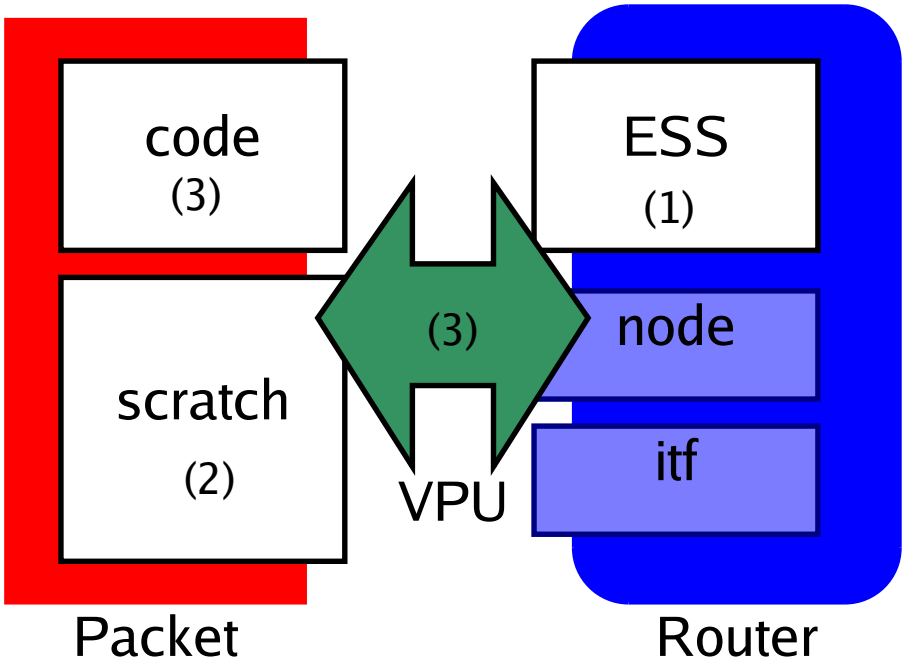
Inside WASP: The Virtual Processing Unit



1. lookup/insert into ESS
2. load/store data from packet's scratchpad
3. interpreted, RISC-inspired **packet bytecode**
4. control opcodes (drop, forward, return)

- tiny interpreter code (4K)
- data and code used 'as is' (no marshalling)
- simple ALU design and compact opcodes

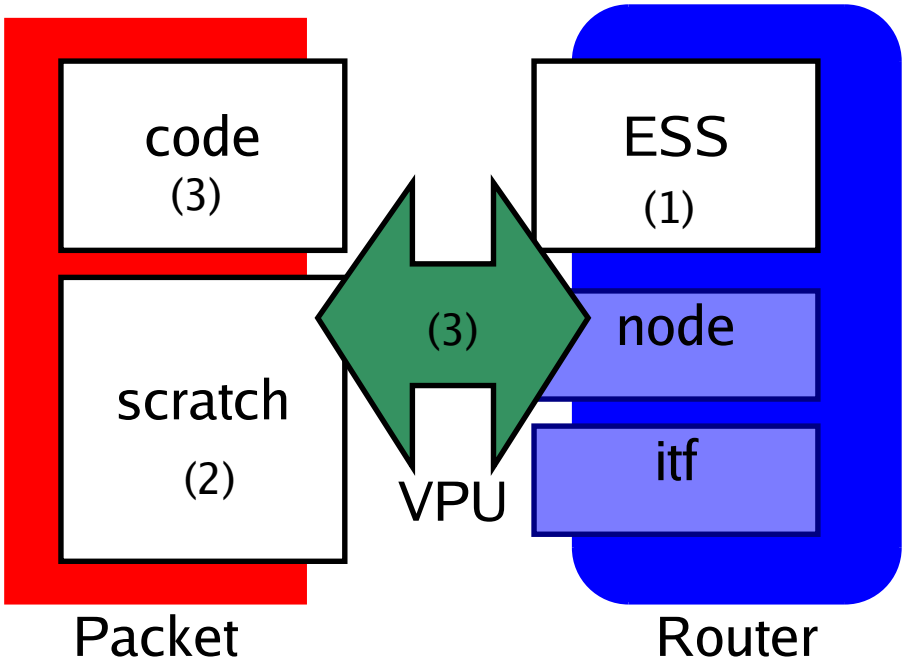
Inside WASP: The Virtual Processing Unit



1. lookup/insert into ESS
2. load/store data from packet's scratchpad
3. interpreted, RISC-inspired packet bytecode
4. **control opcodes** (drop, forward, return)

- tiny interpreter code (4K)
- data and code used 'as is' (no marshalling)
- simple ALU design and compact opcodes

Inside WASP: The Virtual Processing Unit

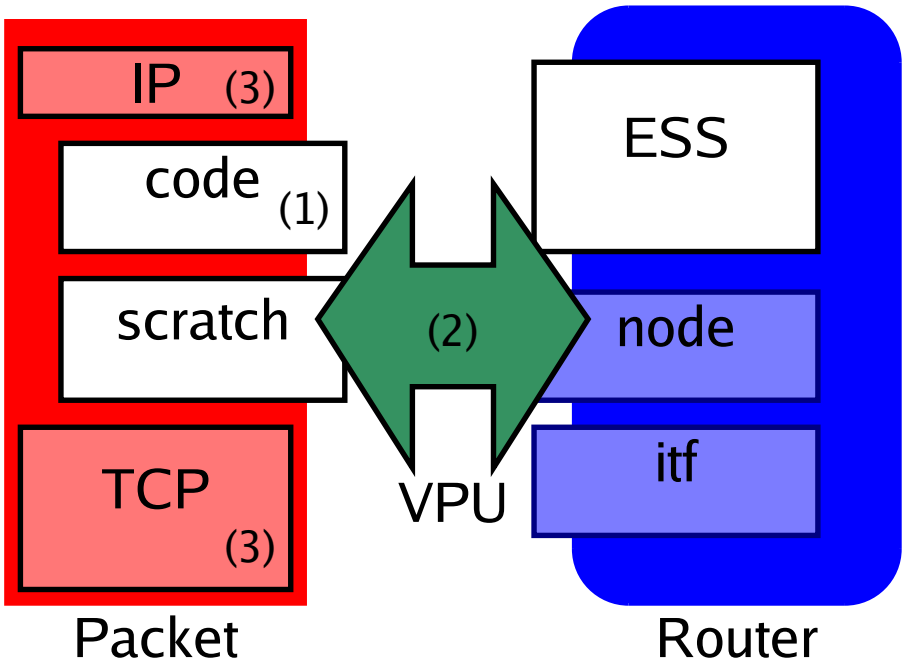


1. lookup/insert into ESS
2. load/store data from packet's scratchpad
3. interpreted, RISC-inspired packet bytecode
4. control opcodes (drop, forward, return)

- tiny interpreter code (4K)
- data and code used 'as is' (no marshalling)
- simple ALU design and compact opcodes

WASP is Safe!

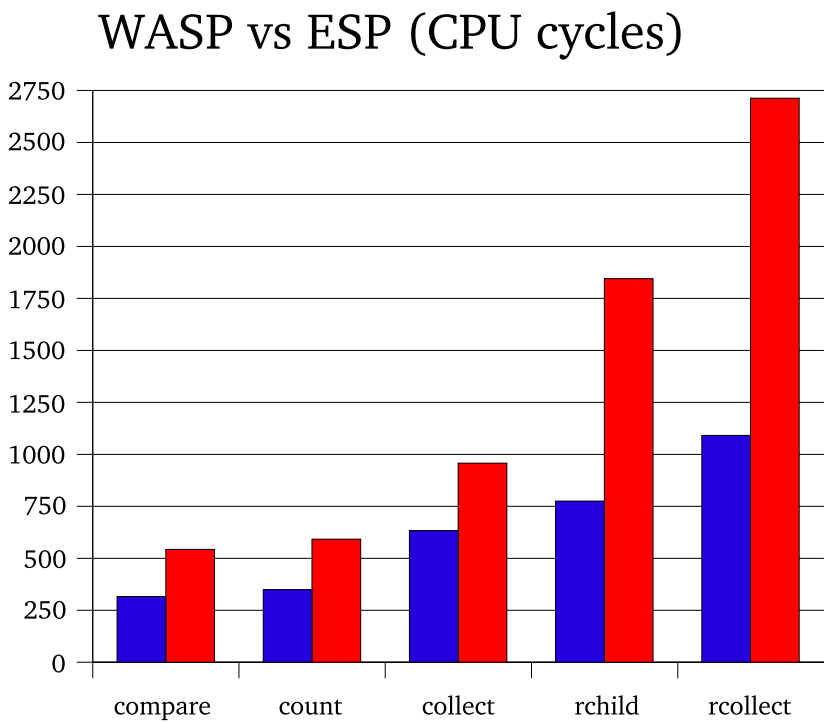
cf. *Practical Active Packets, Jonathan T. Moore*, University of Pennsylvania

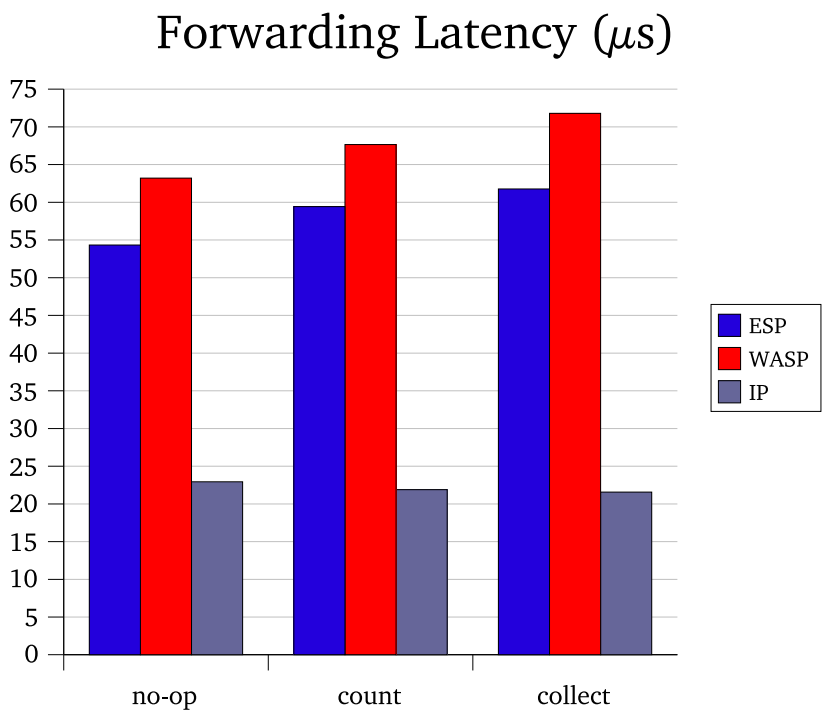


1. no backward jumps
2. no heavy computation
3. transparent to other protocols
4. no packets cloning

So WASP is inherently router-friendly.

VPU Performance

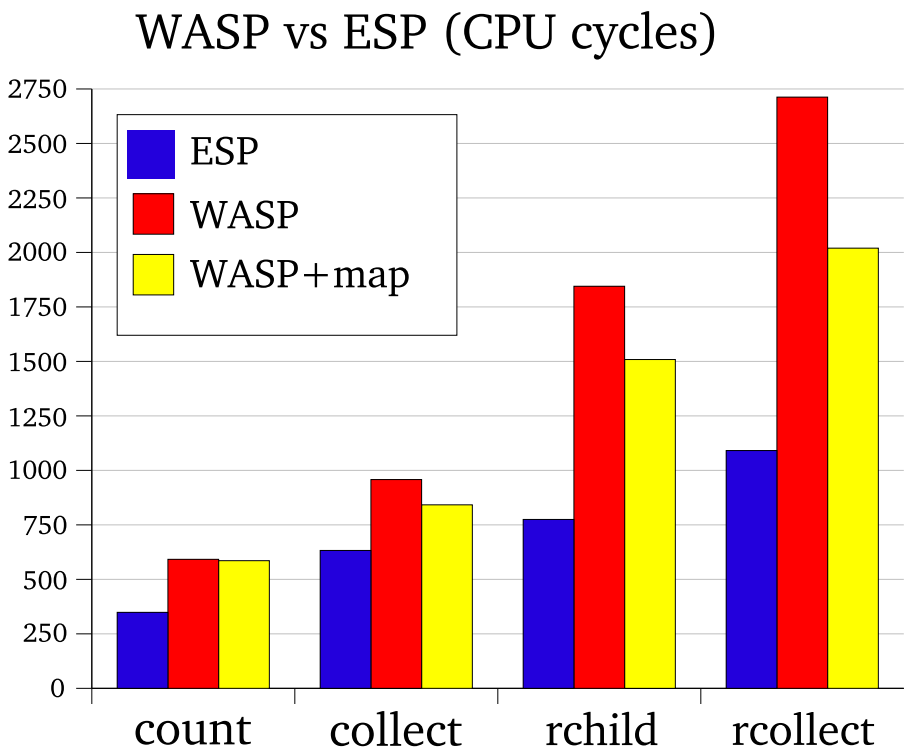




- benchmark on a Pentium Linux router.
- interpretation takes 150% to 250% of native execution time
- only a small part of packet latency (115% overhead)

Mapping Larger Tags for Better Performance

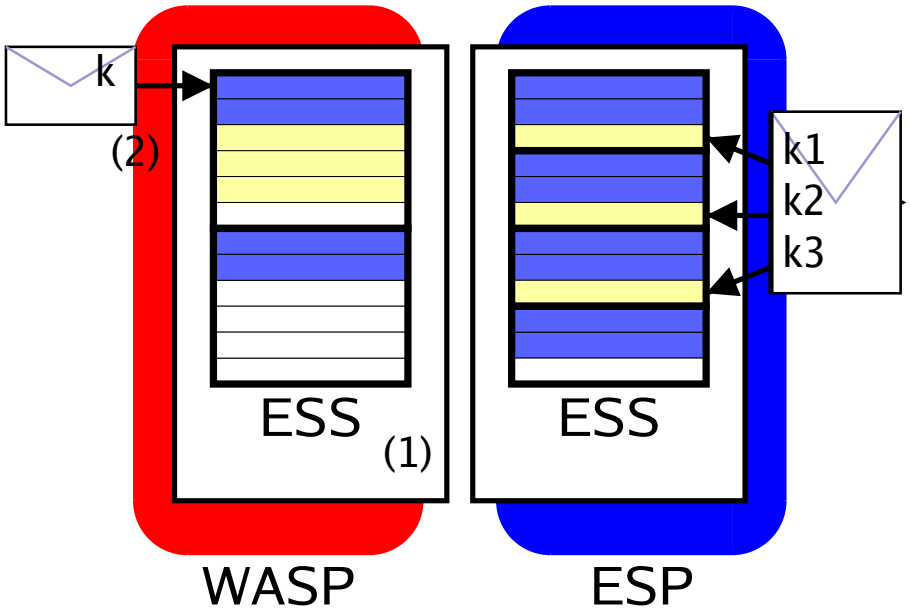
Preformance Improvement



- single key contain all service state
- smaller packets (fewer keys needed)
- faster processing (185% of ESP)

Mapping Larger Tags for Better Performance

How It Works

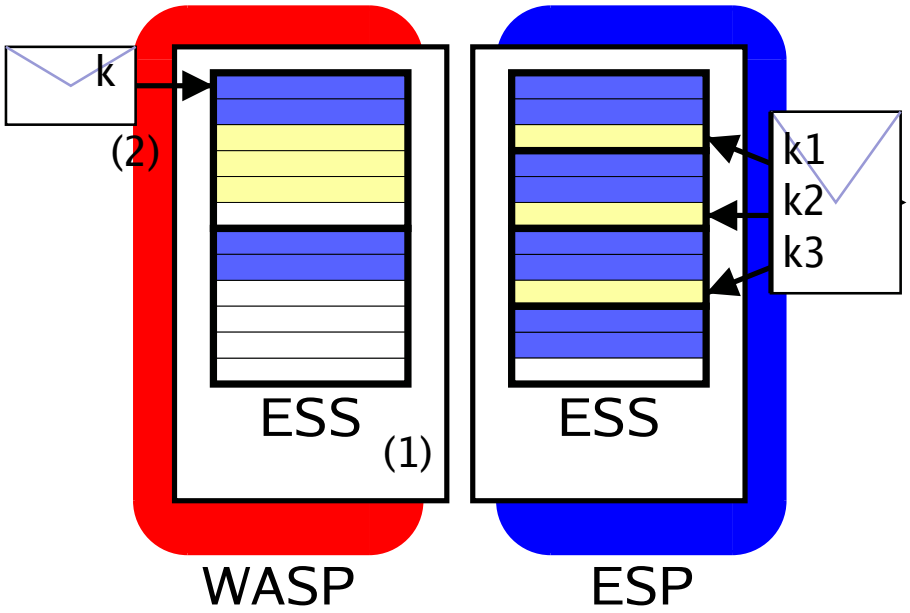


- size up ESS entries to 32 bytes
- lookup ESS only once
- load/store to mapped copy
- write back when done

- good performance expected on IXP (SDRAM latency)
- no memory overhead if at least 2 words per state

Mapping Larger Tags for Better Performance

How It Works

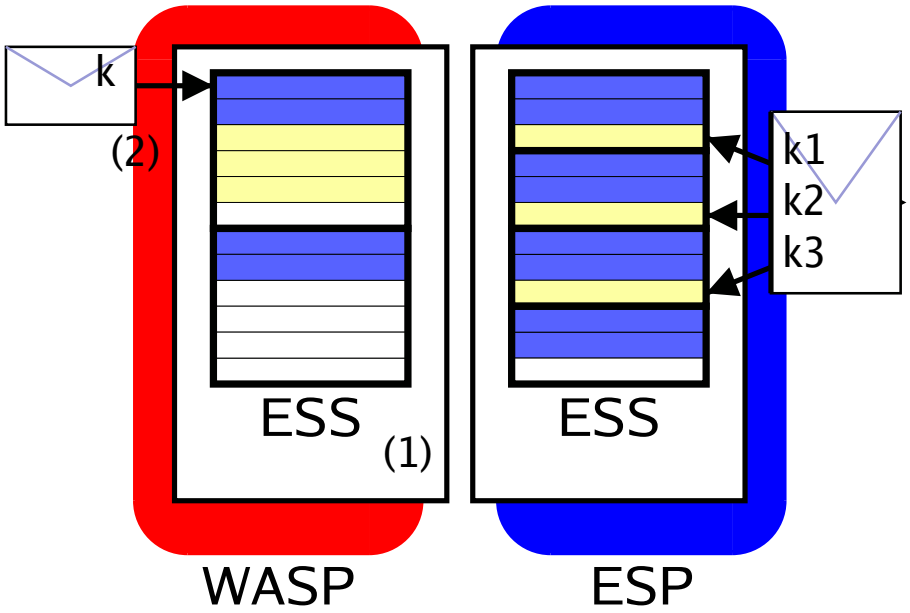


- size up ESS entries to 32 bytes
- lookup ESS only once
- load/store to mapped copy
- write back when done

- good performance expected on IXP (SDRAM latency)
- no memory overhead if at least 2 words per state

Mapping Larger Tags for Better Performance

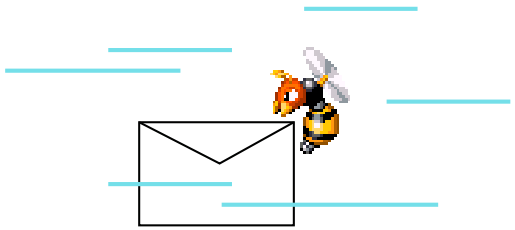
How It Works



- size up ESS entries to 32 bytes
- lookup ESS only once
- load/store to mapped copy
- write back when done

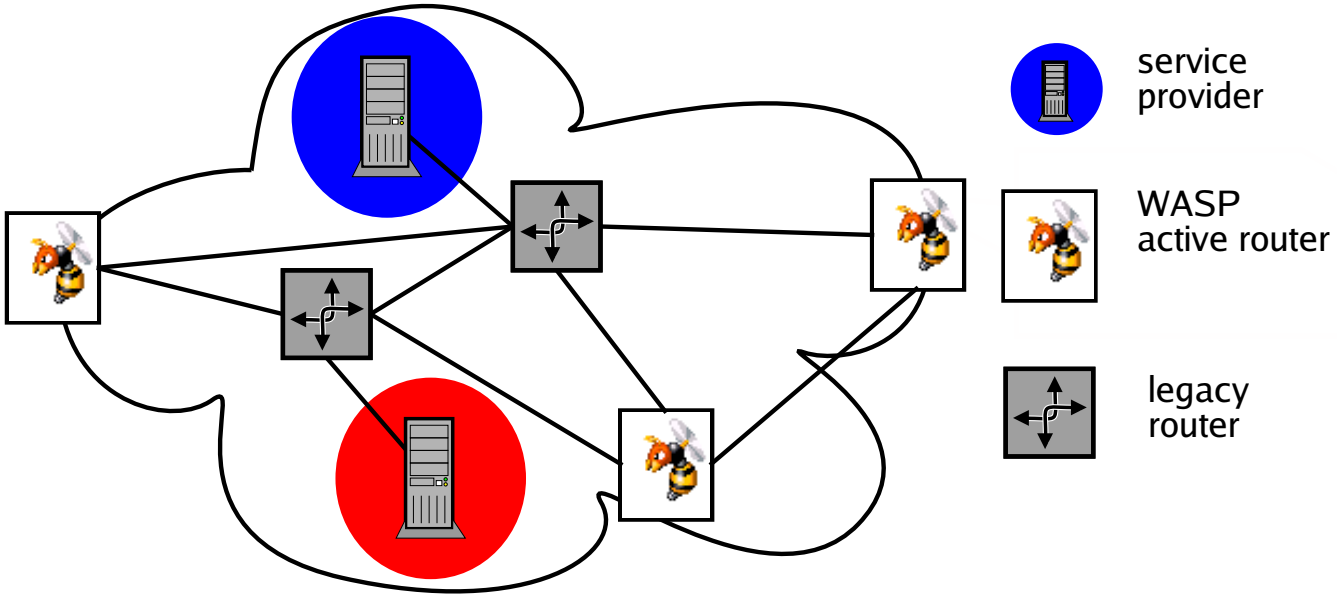
- good performance expected on IXP (SDRAM latency)
- no memory overhead if at least 2 words per state

Discovering Services with WASP



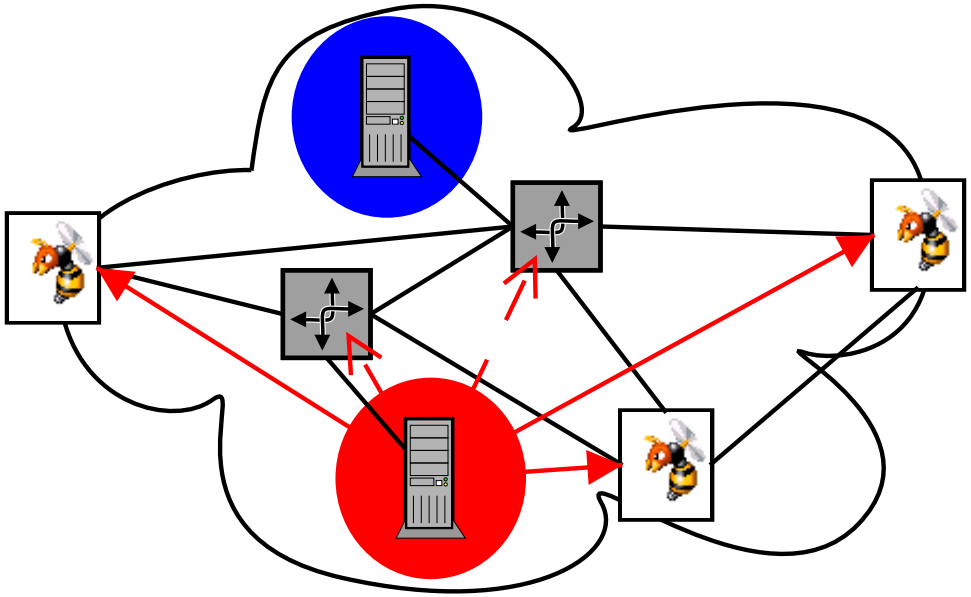
How can WASP help for more complex services ?

MagNet (1): Advertise Service Within the Domain.



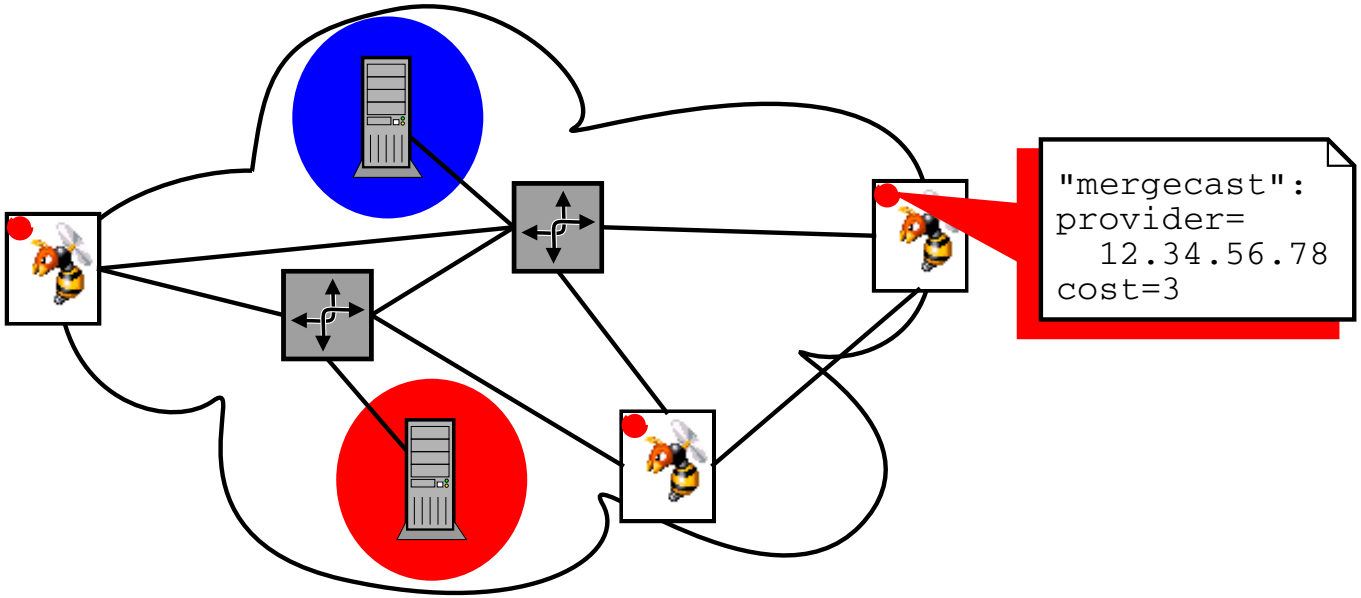
1. providers learn address of local routers,
2. send them active packets ...
3. ... which leave advertisements in active routers
4. active code also select best advertisements when needed

MagNet (1): Advertise Service Within the Domain.



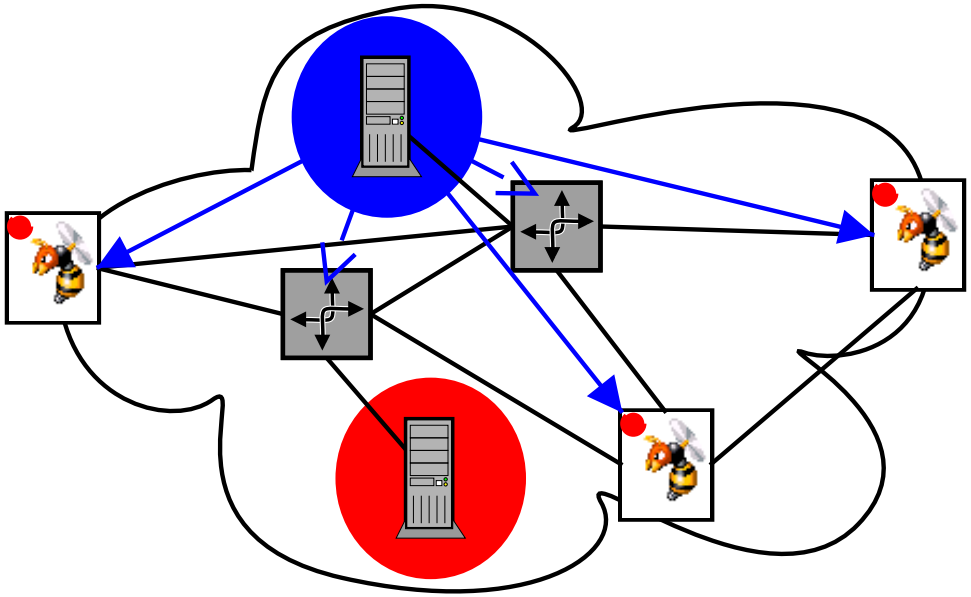
1. providers learn address of local routers,
2. send them active packets ...
3. ... which leave advertisements in active routers
4. active code also select best advertisements when needed

MagNet (1): Advertise Service Within the Domain.



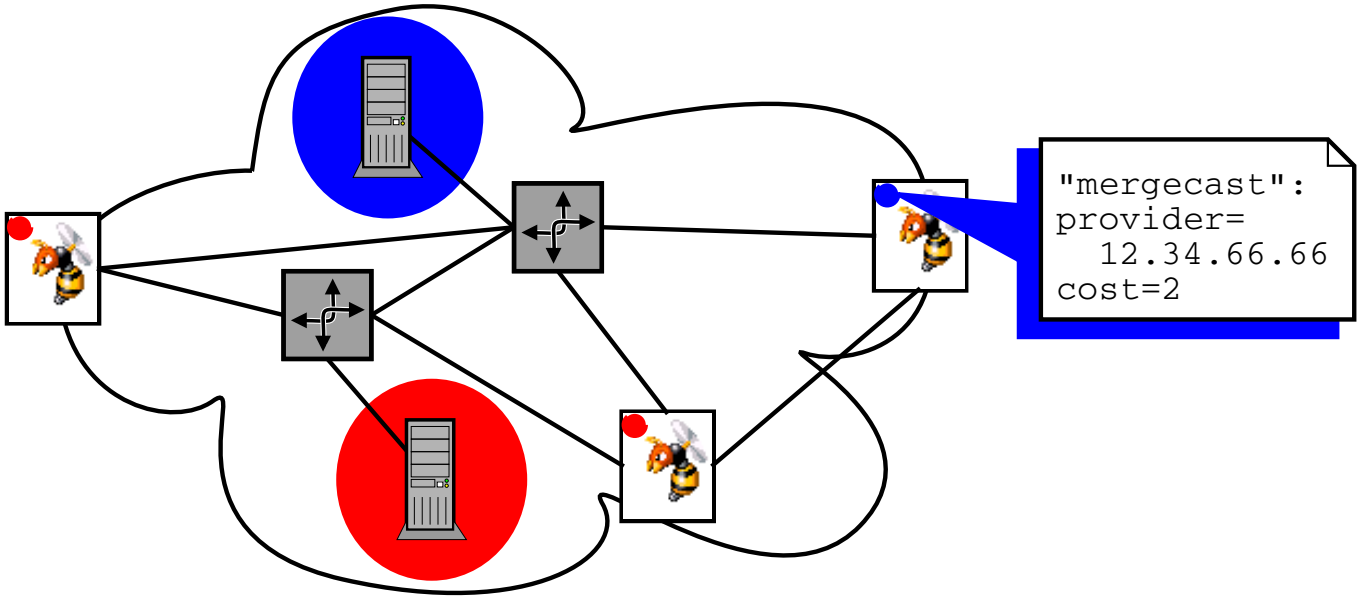
1. providers learn address of local routers,
2. send them active packets ...
3. ... which leave advertisements in active routers
4. active code also select best advertisements when needed

MagNet (1): Advertise Service Within the Domain.



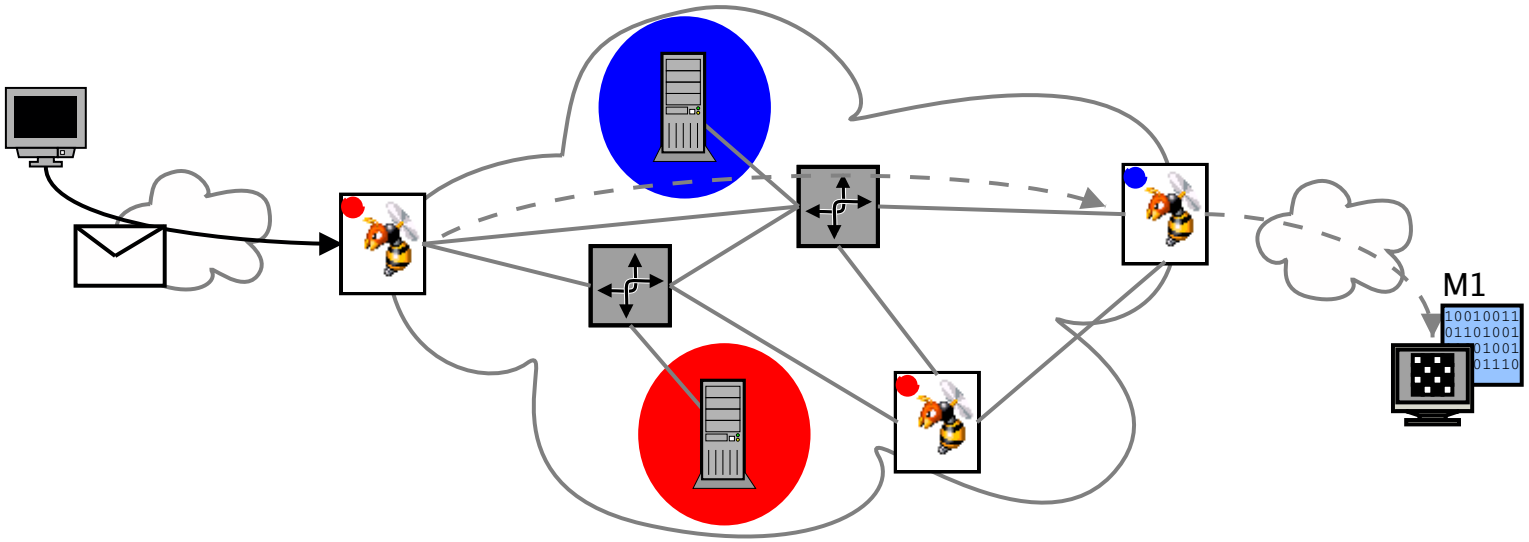
1. providers learn address of local routers,
2. send them active packets ...
3. ... which leave advertisements in active routers
4. active code also select best advertisements when needed

MagNet (1): Advertise Service Within the Domain.



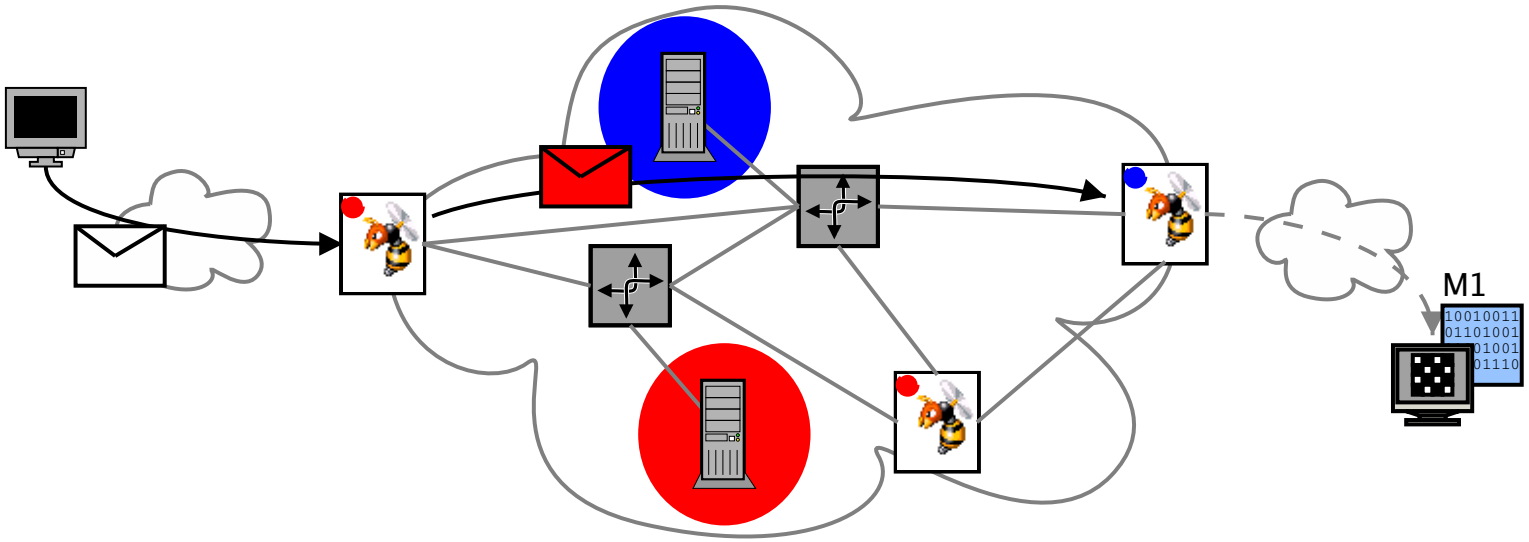
1. providers learn address of local routers,
2. send them active packets ...
3. ... which leave advertisements in active routers
4. active code also select best advertisements when needed

MagNet(2): Look For Service At Connection Setup.



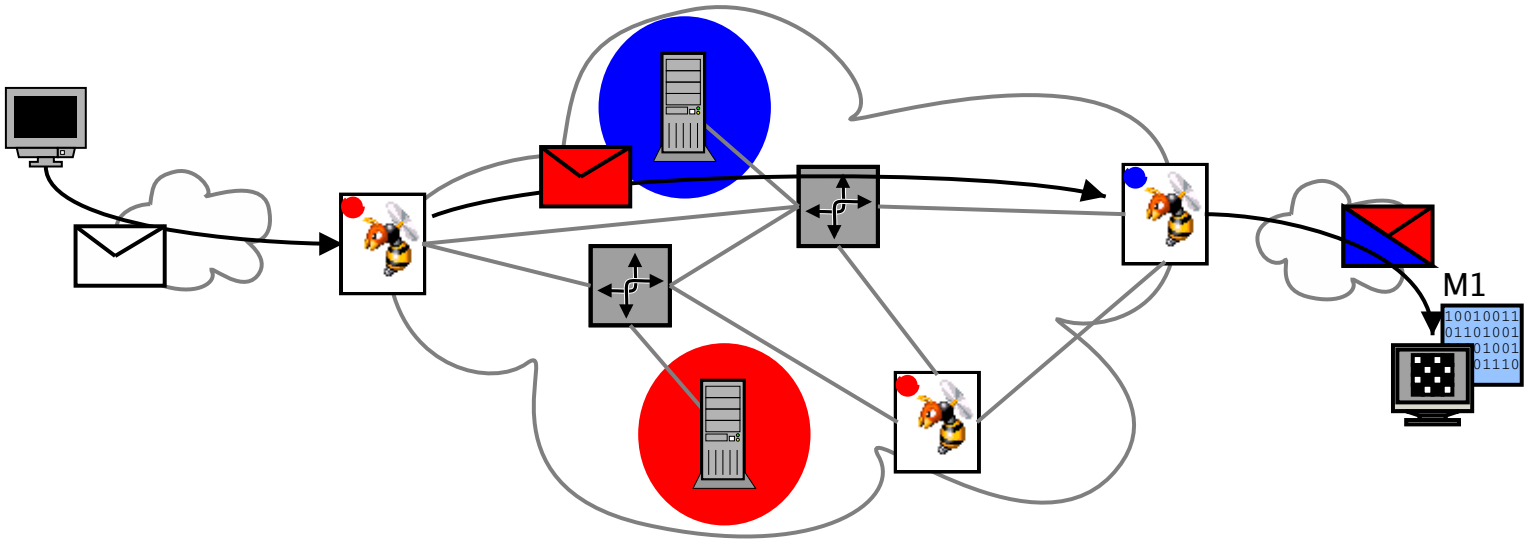
- send a connection request to retrieve results
- active code checks for service availability and store location information in packet
- then let the packet go and store more addresses
- when finally at target, send back the whole list to requestor.

MagNet(2): Look For Service At Connection Setup.



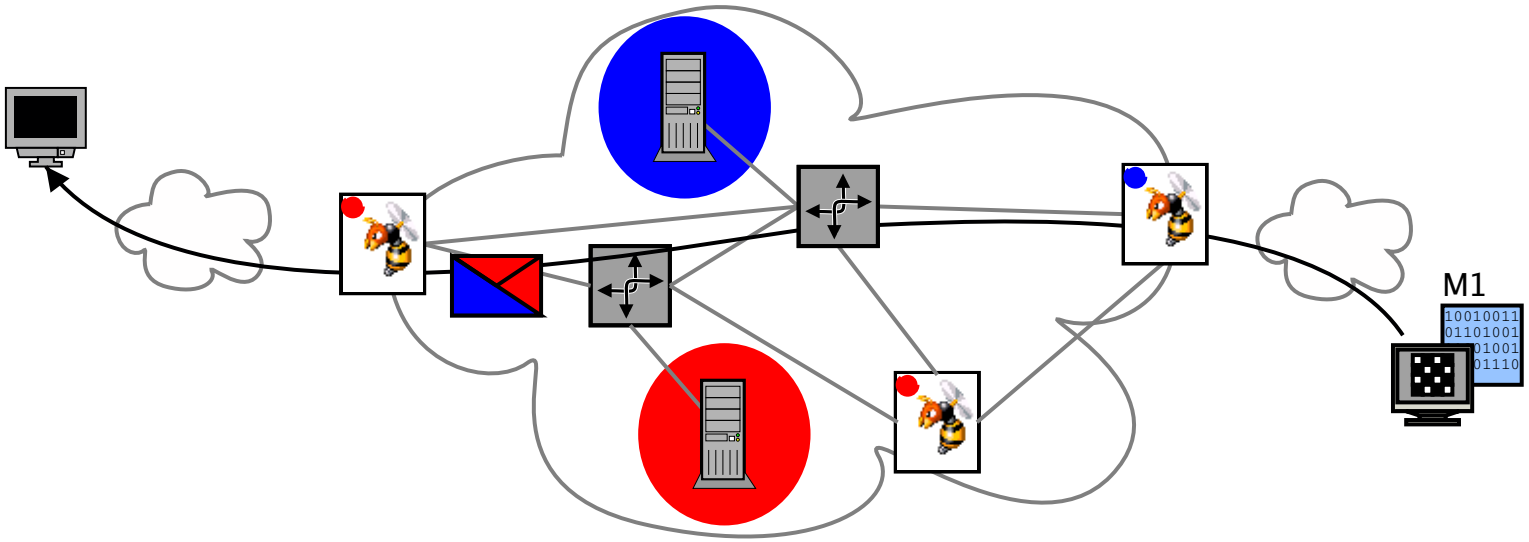
- send a connection request to retrieve results
- active code checks for service availability and store location information in packet
- then let the packet go and store more addresses
- when finally at target, send back the whole list to requestor.

MagNet(2): Look For Service At Connection Setup.



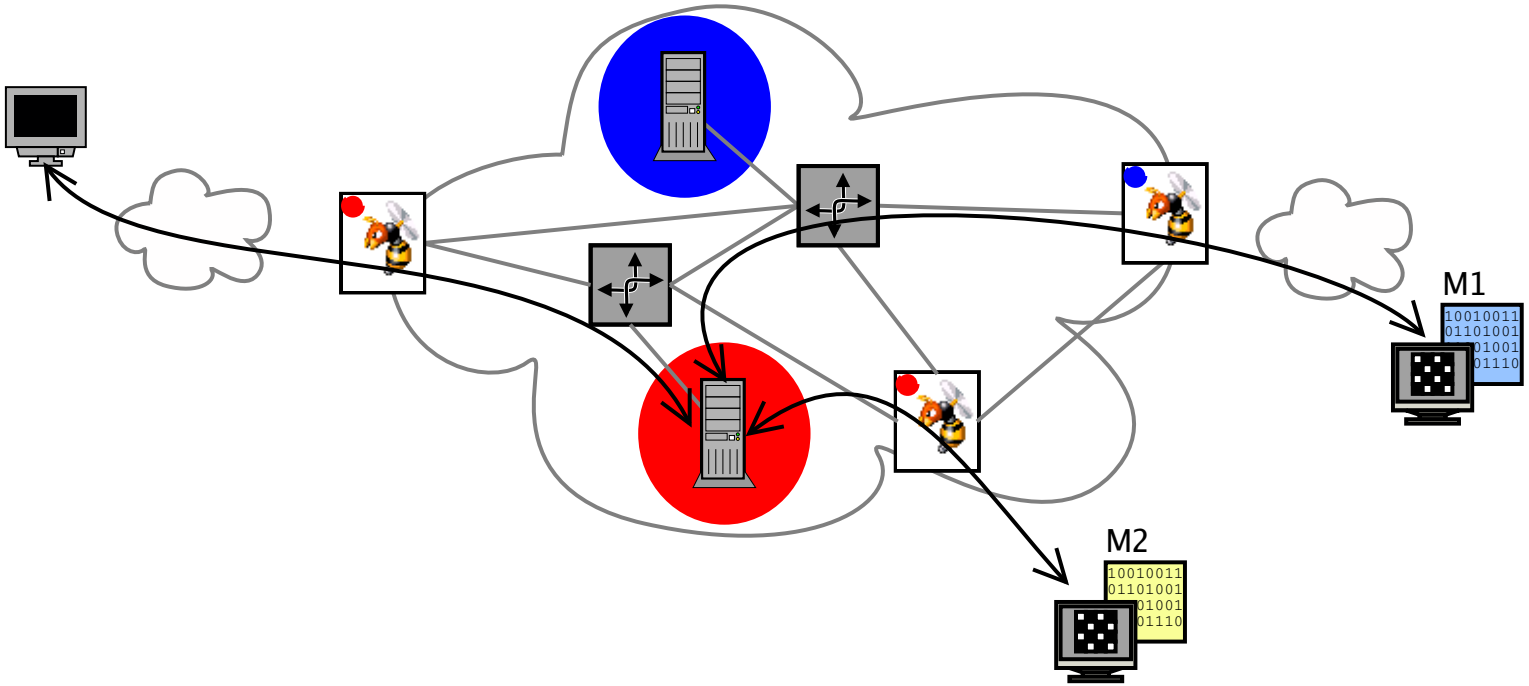
- send a connection request to retrieve results
- active code checks for service availability and store location information in packet
- then let the packet go and store more addresses
- when finally at target, send back the whole list to requestor.

MagNet(2): Look For Service At Connection Setup.



- send a connection request to retrieve results
- active code checks for service availability and store location information in packet
- then let the packet go and store more addresses
- when finally at target, send back the whole list to requestor.

Finally Use The Service.



Client can gather all information required to set up the preferred service provider(s).

Extending Storage Semantics

WASP divides keys space for extended tag semantics:

- **public** assume all keys are random and world-writable (default)
- **protected** only operator can write, anyone can read.
- **private** key is generated by hashing bytecode tag cannot be accessed otherwise.

Well-known protected keys are required for services discovery.

Extending Storage Semantics

WASP divides keys space for extended tag semantics:

public assume all keys are random and world-writable (default)

protected only operator can write, anyone can read.

private key is generated by hashing bytecode tag cannot be accessed otherwise.

Well-known protected keys are required for services discovery.

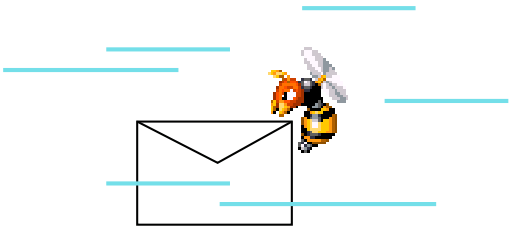
Extending Storage Semantics

WASP divides keys space for extended tag semantics:

- **public** assume all keys are random and world-writable (default)
- **protected** only operator can write, anyone can read.
- **private** key is generated by hashing bytecode tag cannot be accessed otherwise.

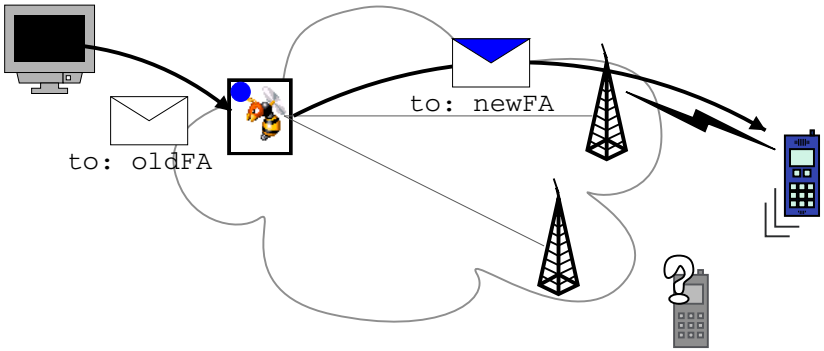
Well-known protected keys are required for services discovery.

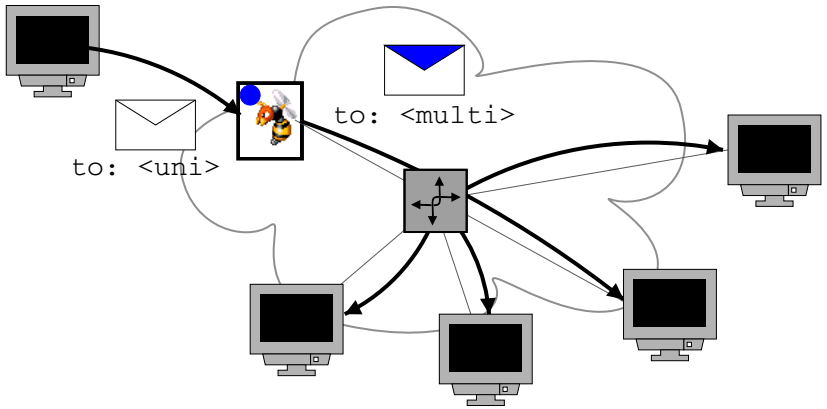
Rerouting



What about changing destination address on the fly ?

World-Friendly Rerouting (?)





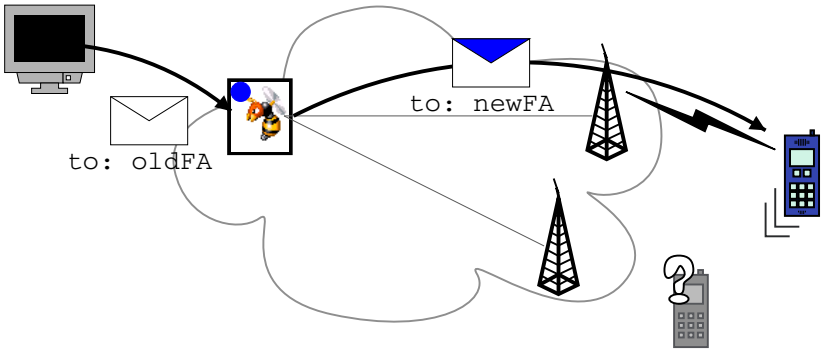
Pros

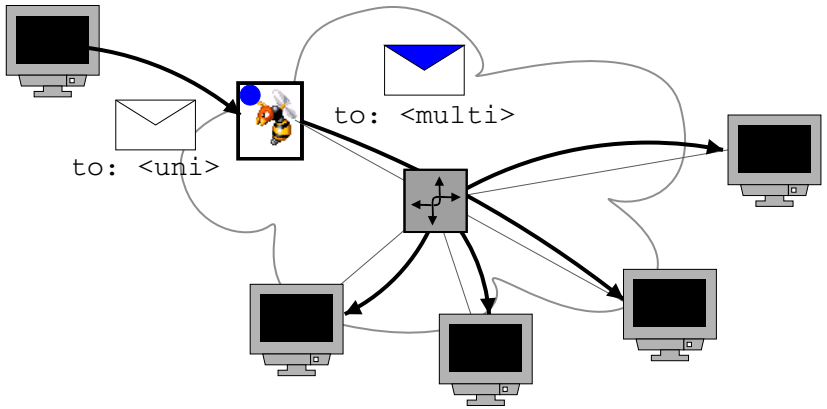
- useful for real-time multimedia
- hybrid multicast/unicast routes

Cons

- multiple IP table lookups
- looping packets ?
- billing ?
- who gets my packets ??

World-Friendly Rerouting (?)





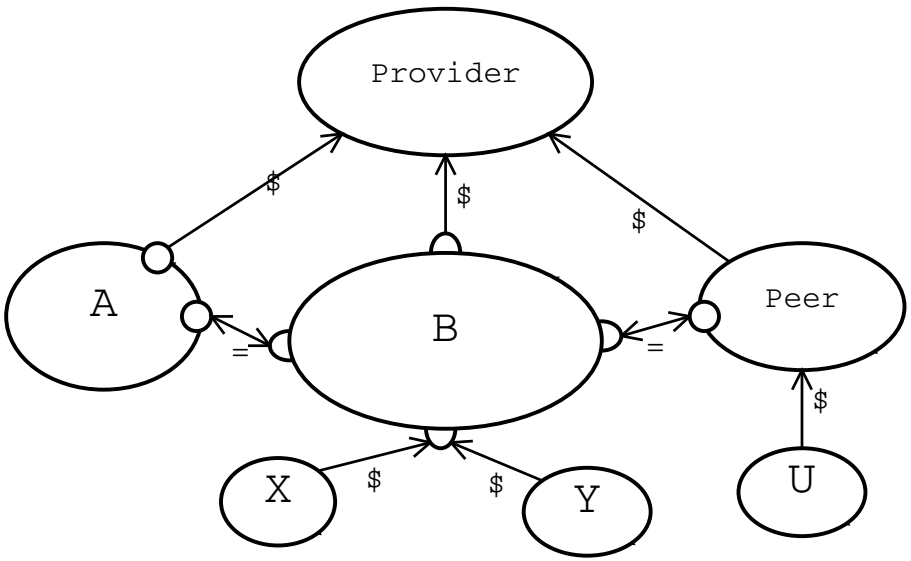
Pros

- useful for real-time multimedia
- hybrid multicast/unicast routes

Cons

- multiple IP table lookups
- looping packets ?
- billing ?
- who gets my packets ??

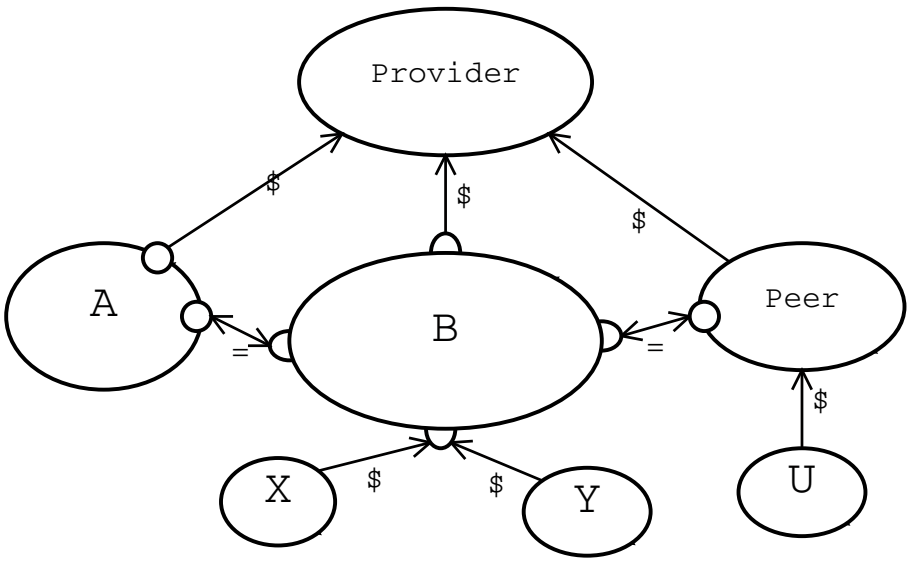
Interdomain Rerouting Made Friendly



- multiple IP table lookups
- looping packets ?
- billing ?

1. Rerouting allowed on ingress/egress interfaces only
2. At egress, ensure new address goes through same interface
3. Ensure guest packets can be rerouted only to clients

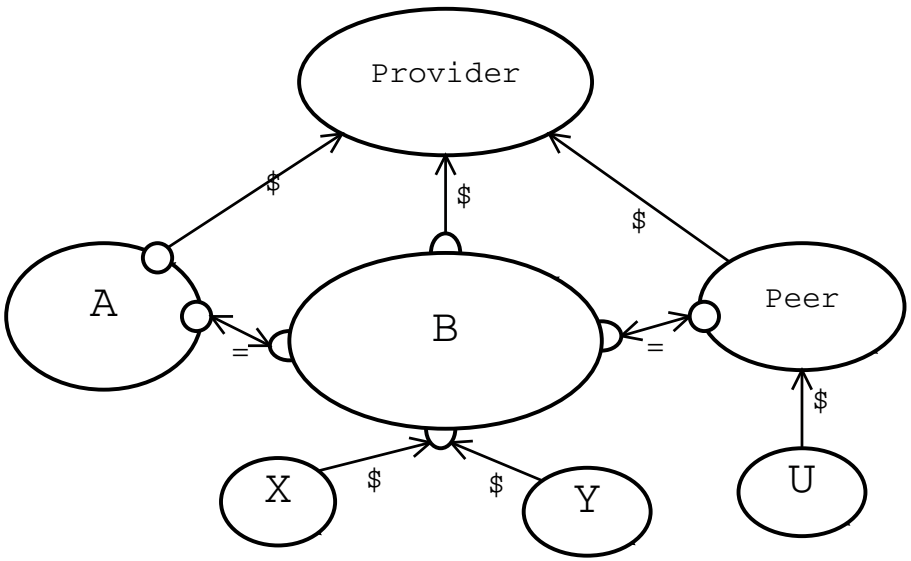
Interdomain Rerouting Made Friendly



- multiple IP table lookups
- looping packets ?
- billing ?

1. Rerouting allowed on ingress/egress interfaces only
2. At egress, ensure new address goes through same interface
3. Ensure guest packets can be rerouted only to clients

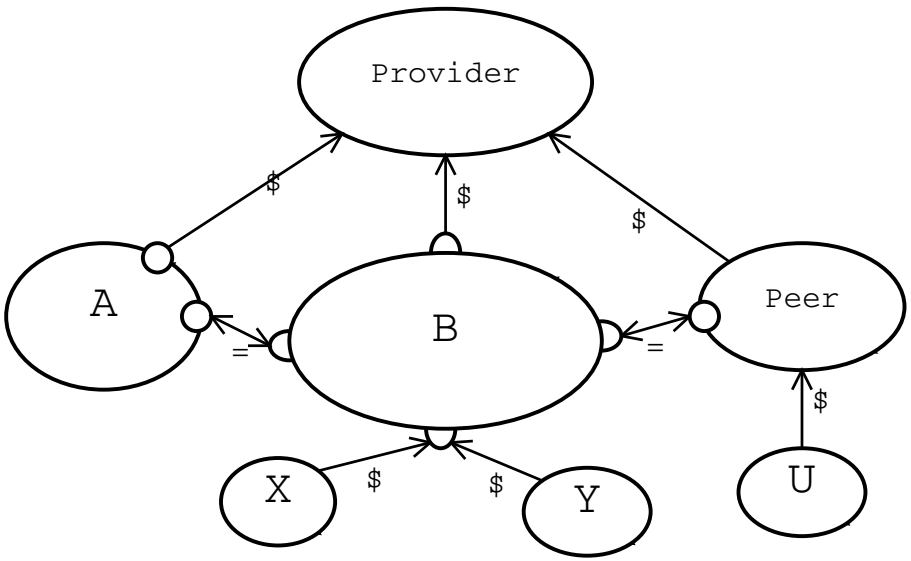
Interdomain Rerouting Made Friendly



- multiple IP table lookups
- looping packets ?
- billing ?

1. Rerouting allowed on ingress/egress interfaces only
2. At egress, ensure new address goes through same interface
3. Ensure guest packets can be rerouted only to clients

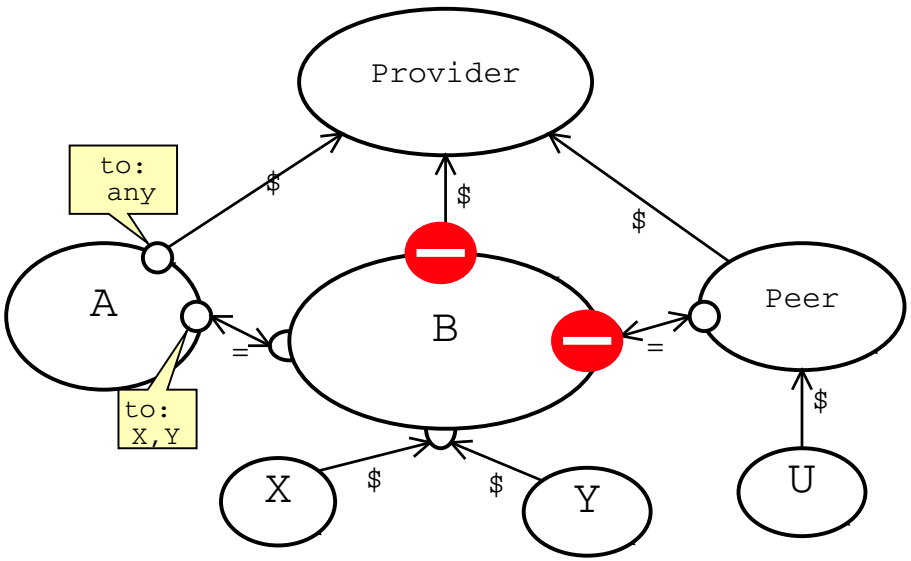
Interdomain Rerouting Made Friendly



- multiple IP table lookups
- looping packets ?
- billing ?

1. Rerouting allowed on ingress/egress interfaces only
2. At egress, ensure new address goes through same interface
3. Ensure guest packets can be rerouted only to clients

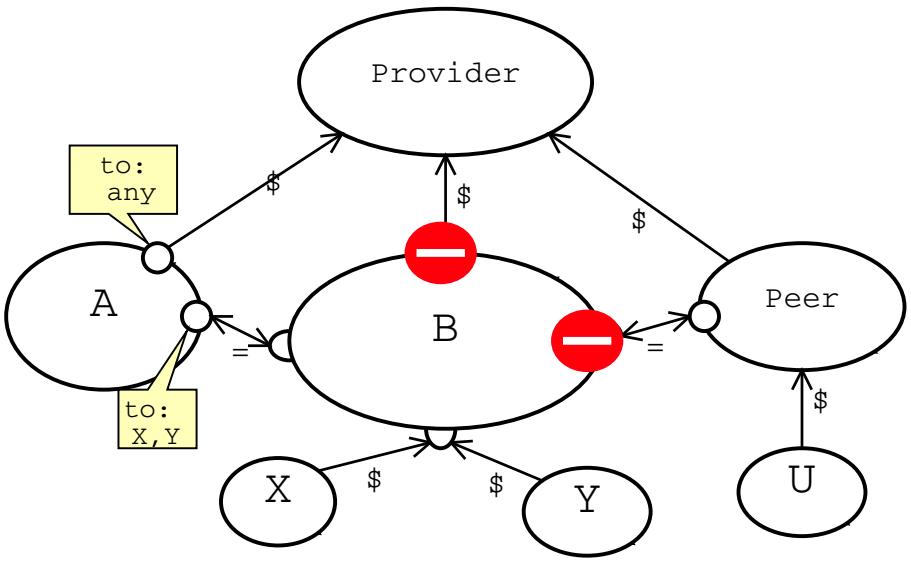
Interdomain Rerouting Made Friendly



- multiple IP table lookups
- looping packets ?
- billing ?

1. Rerouting allowed on ingress/egress interfaces only
2. At egress, ensure new address goes through same interface
3. Ensure guest packets can be rerouted only to clients

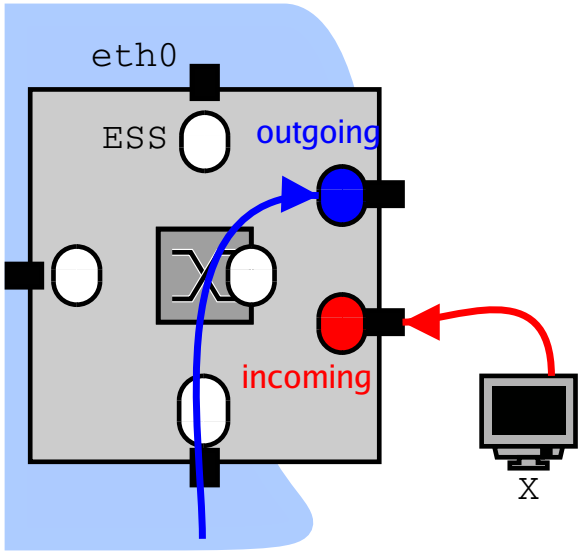
Interdomain Rerouting Made Friendly



- multiple IP table lookups
- looping packets ?
- billing ?

1. Rerouting allowed on ingress/egress interfaces only
2. At egress, ensure new address goes through same interface
3. Ensure guest packets can be rerouted only to clients

Rerouting With Invitations



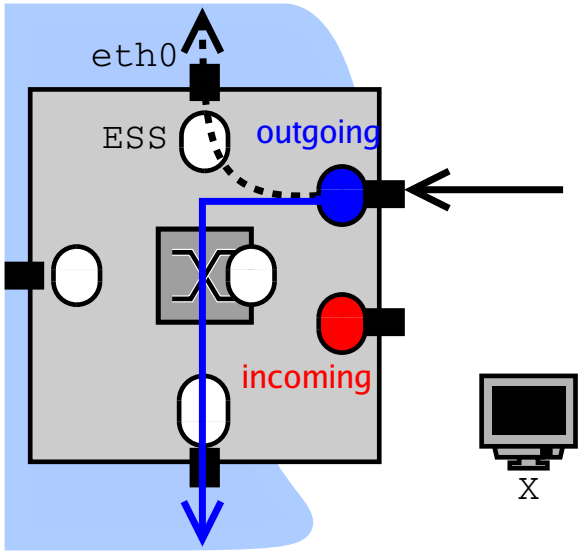
- `invite` store packet source in ESS
- key used can be created only with `invite`
- rerouting can use address of invitations only.

⇒ no packet turnback

⇒ no packet sent on wrong interface

⇒ loop-free if guest packets invite at ingress only.

Rerouting With Invitations



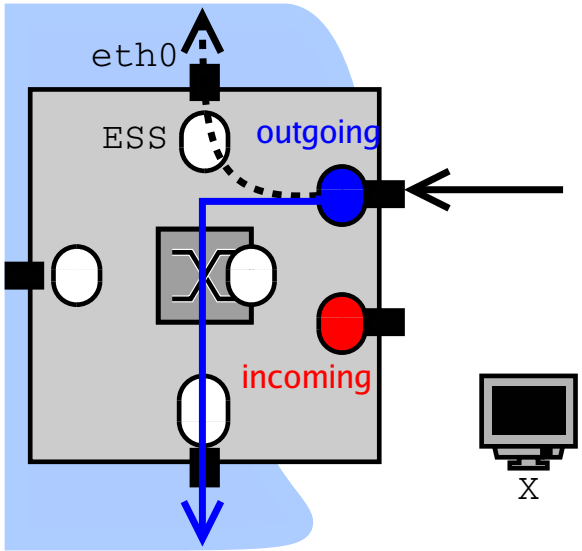
- `invite` store packet source in ESS
- key used can be created only with `invite`
- rerouting can use address of invitations only.

⇒ no packet turnback

⇒ no packet sent on wrong interface

⇒ loop-free if guest packets invite at ingress only.

Rerouting With Invitations



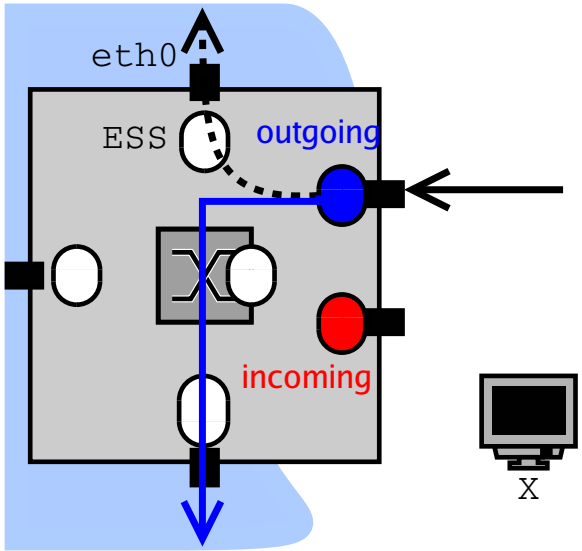
- `invite` store packet source in ESS
- key used can be created only with `invite`
- rerouting can use address of invitations only.

$\Rightarrow$ no packet turnback

$\Rightarrow$ no packet sent on wrong interface

$\Rightarrow$ loop-free if guest packets invite at ingress only.

Rerouting With Invitations



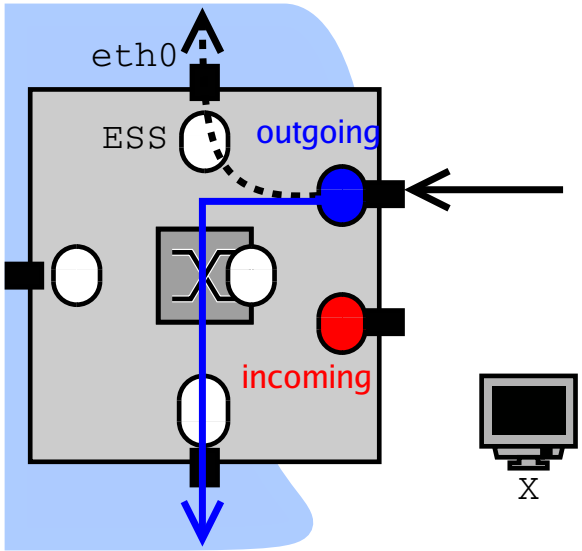
- `invite` store packet source in ESS
- key used can be created only with `invite`
- rerouting can use address of invitations only.

⇒ no packet turnback

⇒ no packet sent on wrong interface

⇒ loop-free if guest packets invite at ingress only.

Rerouting With Invitations



- `invite` store packet source in ESS
- key used can be created only with `invite`
- rerouting can use address of invitations only.

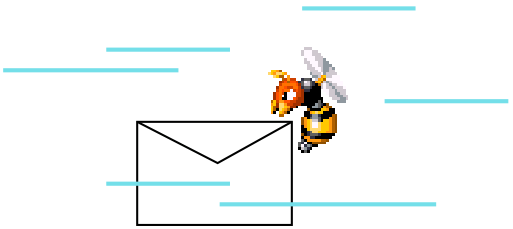
⇒ no packet turnback

⇒ no packet sent on wrong interface

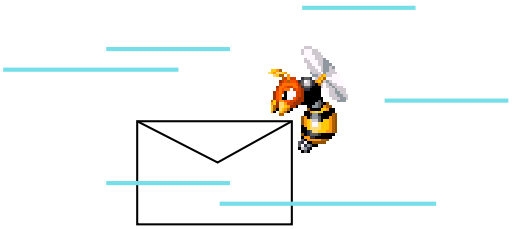
⇒ loop-free if guest packets invite at ingress only.

Conclusions

- Ephemeral State could do more than ESP
- WASP can build simple solutions based on ESS
- Discovery for more complex solutions
- good performance on Pentium
- good hope to be ported to IXP
- Rerouting still needs investigations (DDoS tracking, users' privacy ...)



Questions ?



Anyone ?

Outline

Not Any Packet Can Go Anywhere