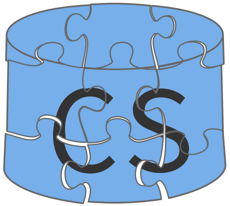


Connecting the Dots: Selective Fragment Recovery in ICNLoWPAN

Martine S. Lenders[*] , Cenk Gündoğan[†], Thomas C. Schmidt[†], Matthias Wählisch[*]

[*]Freie Universität Berlin
{m.lenders,m.waehlisch}@fu-berlin.de

[†]HAW Hamburg
{cenk.guendogan,t.schmidt}@haw-hamburg.de

Outline

Motivation & Background

A Virtual Reassembling Endpoint in ICN

Evaluation of the VREP extension

Conclusion & Outlook

Outline

Motivation & Background

A Virtual Reassembling Endpoint in ICN

Evaluation of the VREP extension

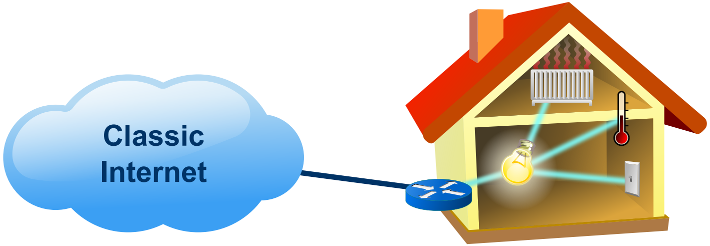
Conclusion & Outlook

Why do we need fragmentation?

Why do we need fragmentation?

PDU: $\sim$1500 bytes

Why do we need fragmentation?

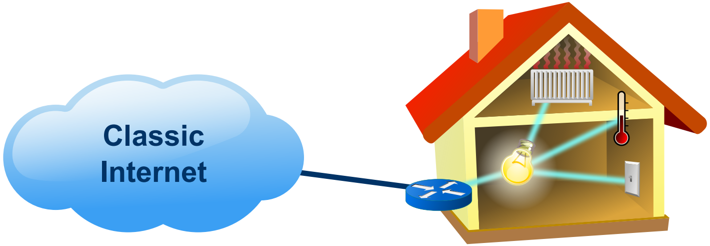


IoT devices

PDU: ∼1500 bytes

PDU: 127 bytes

Why do we need fragmentation?



IoT devices

PDU: ∼1500 bytes

PDU: 127 bytes

6LoWPAN / ICNLOWPAN: Header compression + **Fragmentation**

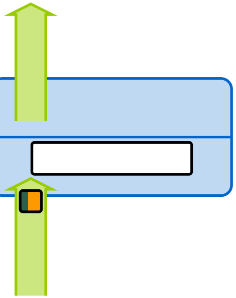
How to forward fragments?

Fragmentation needs: Forwarding information 🟧 + Datagram tag 🟩🟦

How to forward fragments?

Fragmentation needs: Forwarding information 🟧 + Datagram tag 🟩🟦

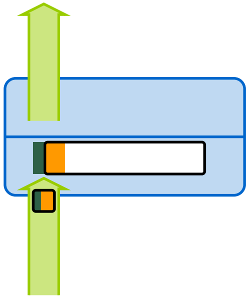
Hop-wise reassembly



How to forward fragments?

Fragmentation needs: Forwarding information 🟧 + Datagram tag 🟩🟦

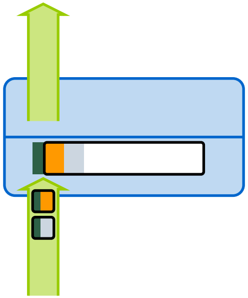
Hop-wise reassembly



How to forward fragments?

Fragmentation needs: Forwarding information 🟧 + Datagram tag 🟩🟦

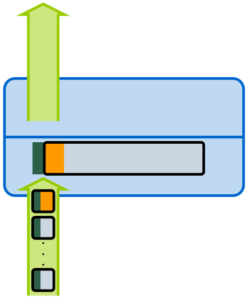
Hop-wise reassembly



How to forward fragments?

Fragmentation needs: Forwarding information 🟧 + Datagram tag 🟩🟦

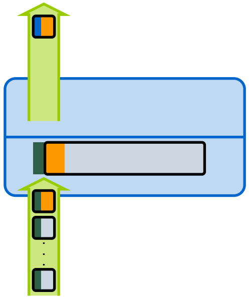
Hop-wise reassembly



How to forward fragments?

Fragmentation needs: Forwarding information 🟧 + Datagram tag 🟩🟦

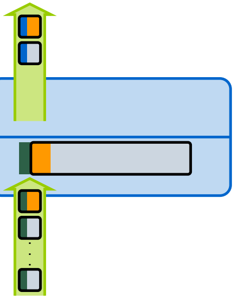
Hop-wise reassembly



How to forward fragments?

Fragmentation needs: Forwarding information 🟧 + Datagram tag 🟩🟦

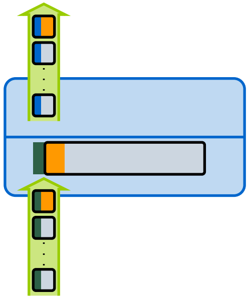
Hop-wise reassembly



How to forward fragments?

Fragmentation needs: Forwarding information 🟧 + Datagram tag 🟩🟦

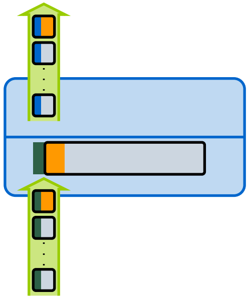
Hop-wise reassembly



How to forward fragments?

Fragmentation needs: Forwarding information 🟧 + Datagram tag 🟩🟦

Hop-wise reassembly



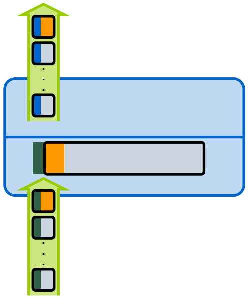
Wastes Memory

→ Forwarding bottlenecks

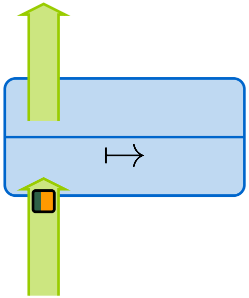
How to forward fragments?

Fragmentation needs: Forwarding information 🟧 + Datagram tag 🟩🟦

Hop-wise reassembly



Direct fragment forwarding



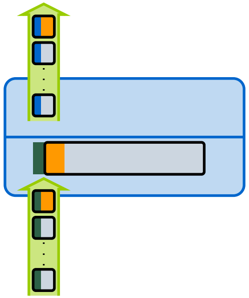
Wastes Memory

→ Forwarding bottlenecks

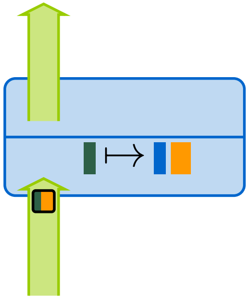
How to forward fragments?

Fragmentation needs: Forwarding information 🟧 + Datagram tag 🟩🟦

Hop-wise reassembly



Direct fragment forwarding



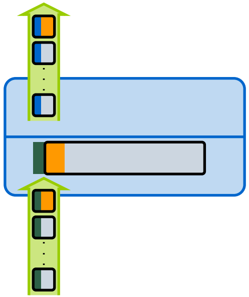
Wastes Memory

→ Forwarding bottlenecks

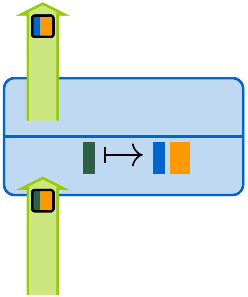
How to forward fragments?

Fragmentation needs: Forwarding information 🟧 + Datagram tag 🟩🟦

Hop-wise reassembly



Direct fragment forwarding



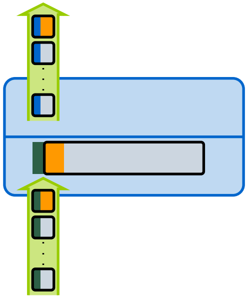
Wastes Memory

→ Forwarding bottlenecks

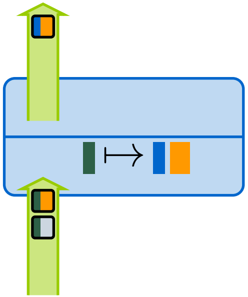
How to forward fragments?

Fragmentation needs: Forwarding information 🟧 + Datagram tag 🟩🟦

Hop-wise reassembly



Direct fragment forwarding



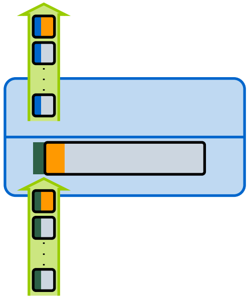
Wastes Memory

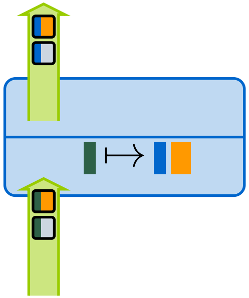
→ Forwarding bottlenecks

How to forward fragments?

Fragmentation needs: Forwarding information 🟧 + Datagram tag 🟩🟦

Hop-wise reassembly | **Direct fragment forwarding**





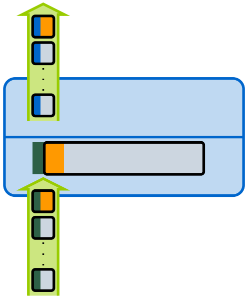
Wastes Memory

→ Forwarding bottlenecks

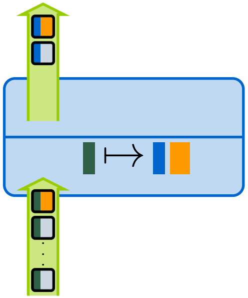
How to forward fragments?

Fragmentation needs: Forwarding information 🟧 + Datagram tag 🟩🟦

Hop-wise reassembly



Direct fragment forwarding



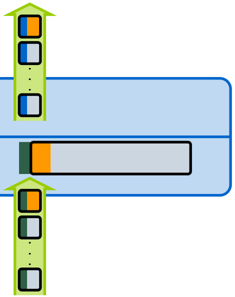
Wastes Memory

$\rightarrow$ Forwarding bottlenecks

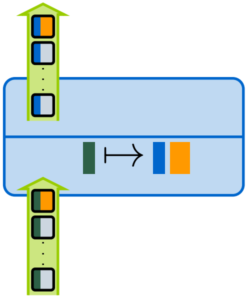
How to forward fragments?

Fragmentation needs: Forwarding information 🟧 + Datagram tag 🟩🟦

Hop-wise reassembly



Direct fragment forwarding



Wastes Memory
→ Forwarding bottlenecks

Minimal memory usage

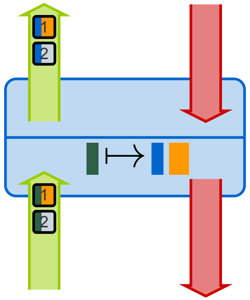
How to recover fragments?

Loosing one fragment requires resending of whole datagram! Solution: Use selective ACK

How to recover fragments?

Loosing one fragment requires resending of whole datagram! Solution: Use selective ACK

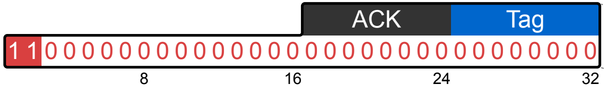
Selective Fragment Recovery



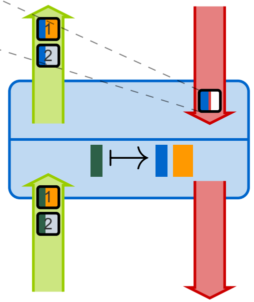
(ACK window size = 2)

How to recover fragments?

Loosing one fragment requires resending of whole datagram! Solution: Use selective ACK



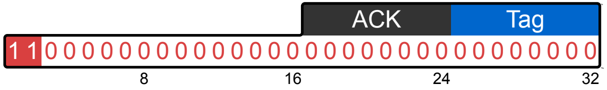
Selective Fragment Recovery



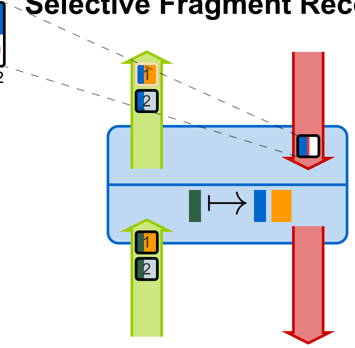
(ACK window size = 2)

How to recover fragments?

Loosing one fragment requires resending of whole datagram! Solution: Use selective ACK



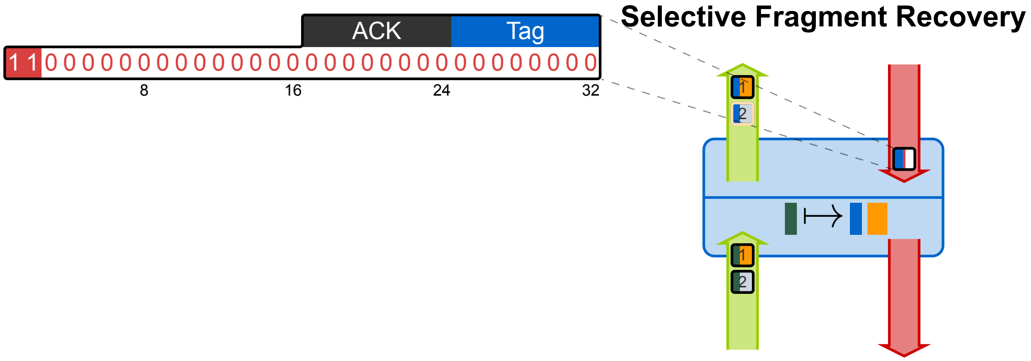
Selective Fragment Recovery



(ACK window size = 2)

How to recover fragments?

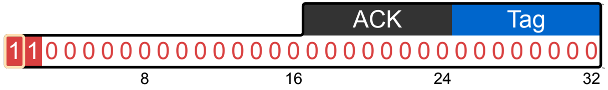
Loosing one fragment requires resending of whole datagram! Solution: Use selective ACK



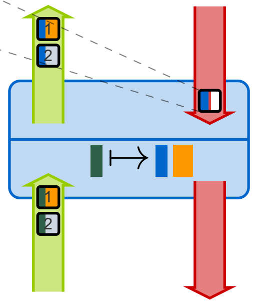
(ACK window size = 2)

How to recover fragments?

Loosing one fragment requires resending of whole datagram! Solution: Use selective ACK



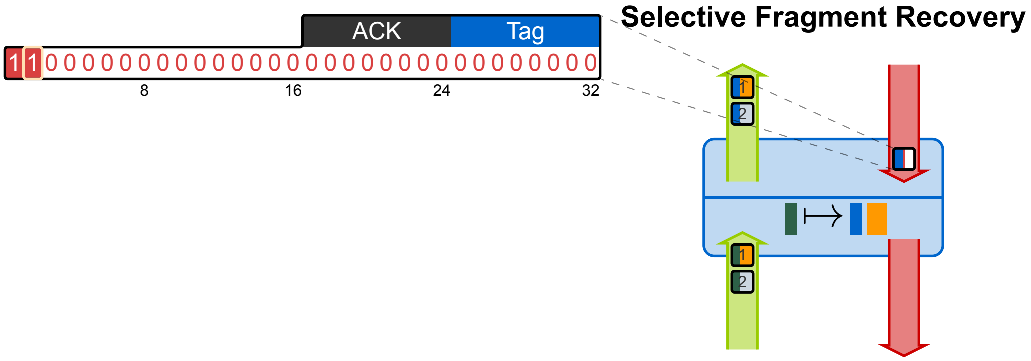
Selective Fragment Recovery



(ACK window size = 2)

How to recover fragments?

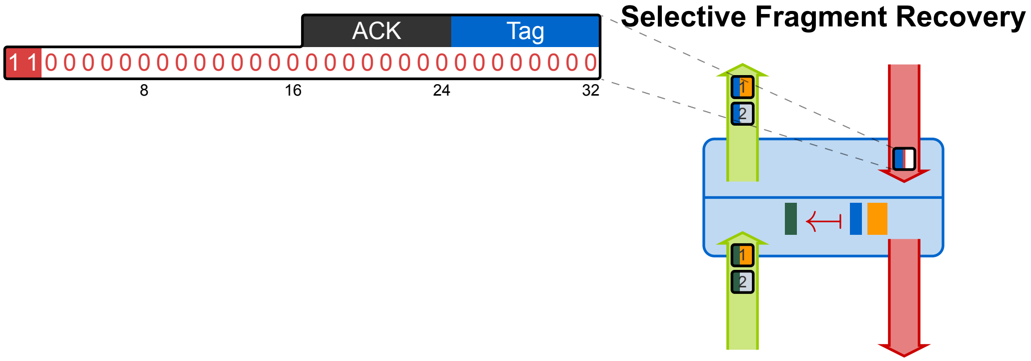
Loosing one fragment requires resending of whole datagram! Solution: Use selective ACK



(ACK window size = 2)

How to recover fragments?

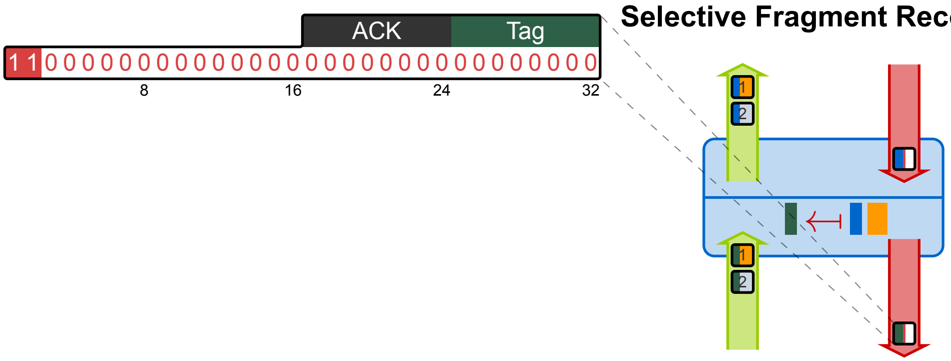
Loosing one fragment requires resending of whole datagram! Solution: Use selective ACK



(ACK window size = 2)

How to recover fragments?

Loosing one fragment requires resending of whole datagram! Solution: Use selective ACK

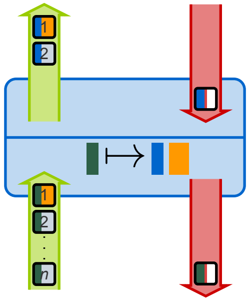


(ACK window size = 2)

How to recover fragments?

Loosing one fragment requires resending of whole datagram! Solution: Use selective ACK

Selective Fragment Recovery

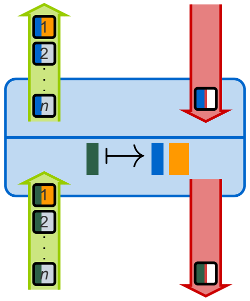


(ACK window size = 2)

How to recover fragments?

Loosing one fragment requires resending of whole datagram! Solution: Use selective ACK

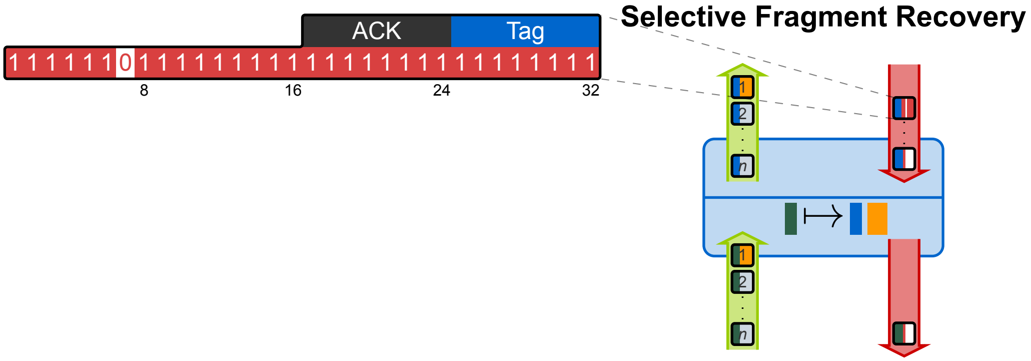
Selective Fragment Recovery



(ACK window size = 2)

How to recover fragments?

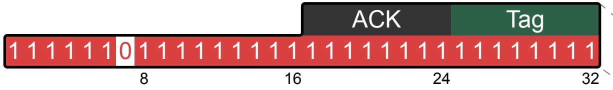
Loosing one fragment requires resending of whole datagram! Solution: Use selective ACK



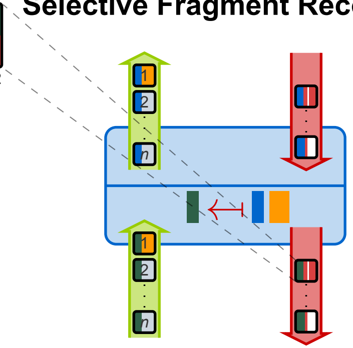
(ACK window size = 2)

How to recover fragments?

Loosing one fragment requires resending of whole datagram! Solution: Use selective ACK



Selective Fragment Recovery

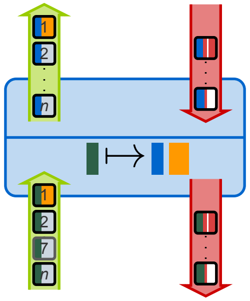


(ACK window size = 2)

Design of the Virtual Reassembling Endpoint

Loosing one fragment requires resending of whole datagram! Solution: Use selective ACK

Selective Fragment Recovery

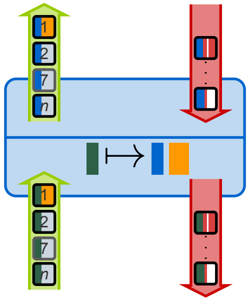


(ACK window size = 2)

Design of the Virtual Reassembling Endpoint

Loosing one fragment requires resending of whole datagram! Solution: Use selective ACK

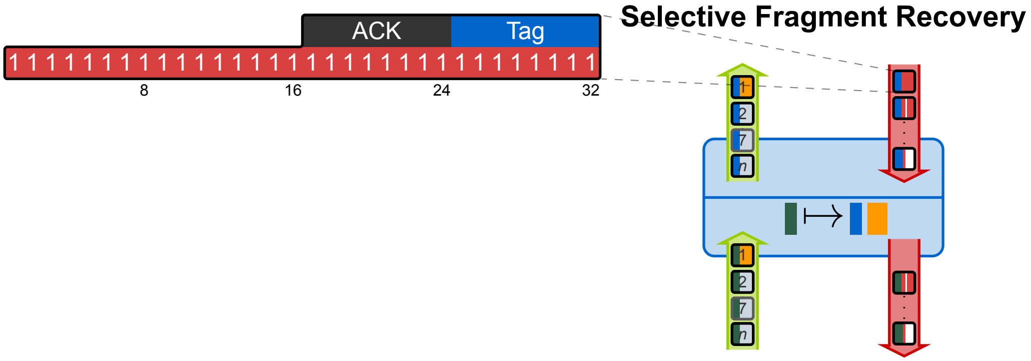
Selective Fragment Recovery



(ACK window size = 2)

Design of the Virtual Reassembling Endpoint

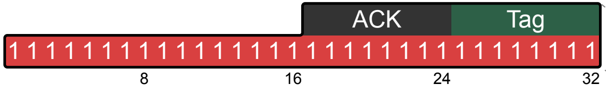
Loosing one fragment requires resending of whole datagram! Solution: Use selective ACK



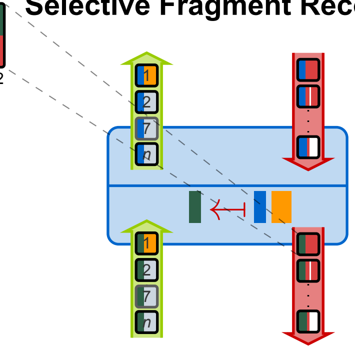
(ACK window size = 2)

Design of the Virtual Reassembling Endpoint

Loosing one fragment requires resending of whole datagram! Solution: Use selective ACK



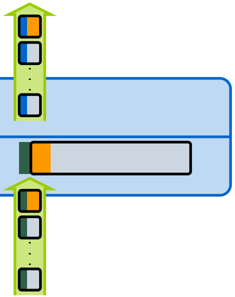
Selective Fragment Recovery



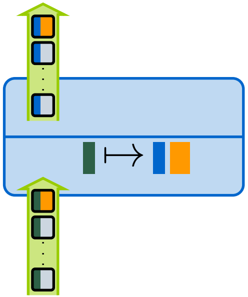
(ACK window size = 2)

Forwarding fragments with ICN Content Store (CS)

Hop-wise reassembly



Direct fragment forwarding



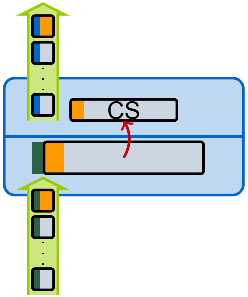
Wastes Memory

→ Forwarding bottlenecks

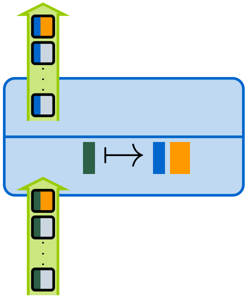
Minimal memory usage

Forwarding fragments with ICN Content Store (CS)

Hop-wise reassembly



Direct fragment forwarding

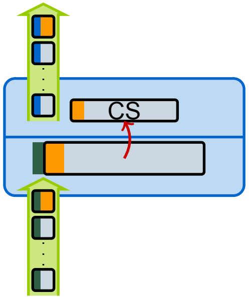


Full content chunk served to ICN

Minimal memory usage

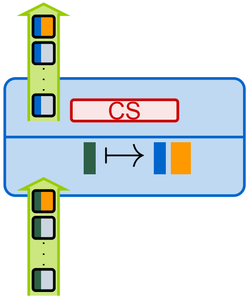
Forwarding fragments with ICN Content Store (CS)

Hop-wise reassembly



Full content chunk served to ICN

Direct fragment forwarding



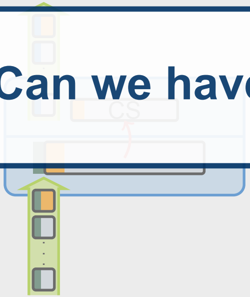
CS never gets complete datagram

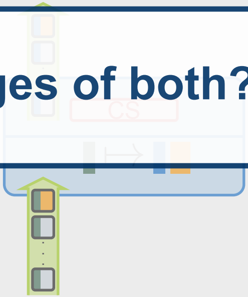
→ ICN unable to cache content chunks

Forwarding fragments with ICN Content Store (CS)

Hop-wise reassembly

Direct fragment forwarding





Can we have the advantages of both?

Full content chunk served to ICN

Minimal memory usage

Outline

Motivation & Background

A Virtual Reassembling Endpoint in ICN

Evaluation of the VREP extension

Conclusion & Outlook

Collecting Requirements

- Full content chunk is served to ICN forwarders

Collecting Requirements

- Full content chunk is served to ICN forwarders
- Safe memory

Collecting Requirements

- Full content chunk is served to ICN forwarders
- Safe memory
    - re-use mostly existing data structures

Collecting Requirements

- Full content chunk is served to ICN forwarders
- Safe memory
  - re-use mostly existing data structures
- Standard compliancy

Collecting Requirements

- Full content chunk is served to ICN forwarders
- Safe memory
  - re-use mostly existing data structures
- Standard compliancy
  - Transparent to the network

Collecting Requirements

- Full content chunk is served to ICN forwarders
- Safe memory
  - re-use mostly existing data structures
- Standard compliancy
  - Transparent to the network
  - Cached content chunk behaves like any other content chunk

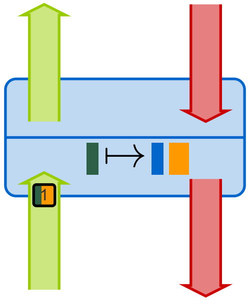
Collecting Requirements

- Full content chunk is served to ICN forwarders
- Safe memory
  - re-use mostly existing data structures
- Standard compliancy
  - Transparent to the network
  - Cached content chunk behaves like any other content chunk
  - Prevent content poisoning with incomplete content chunks

Design of the Virtual Reassembling Endpoint

Reassemble tentatively in CS *en-route*

Selective Fragment Recovery

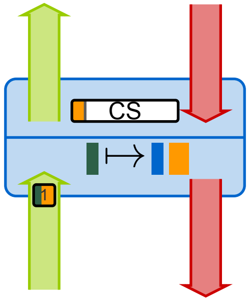


(ACK window size = 2)

Design of the Virtual Reassembling Endpoint

Reassemble tentatively in CS *en-route*

Selective Fragment Recovery

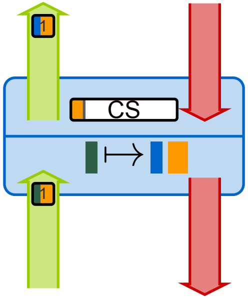


(ACK window size = 2)

Design of the Virtual Reassembling Endpoint

Reassemble tentatively in CS *en-route*

Selective Fragment Recovery

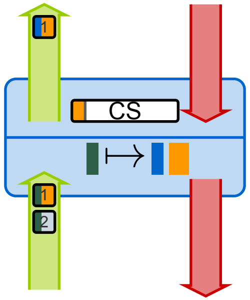


(ACK window size = 2)

Design of the Virtual Reassembling Endpoint

Reassemble tentatively in CS *en-route*

Selective Fragment Recovery

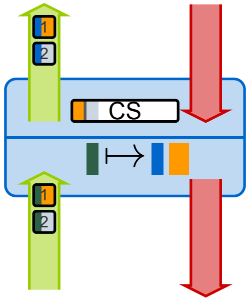


(ACK window size = 2)

Design of the Virtual Reassembling Endpoint

Reassemble tentatively in CS *en-route*

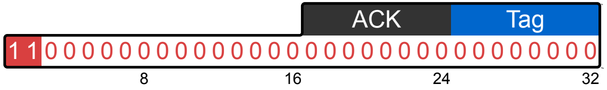
Selective Fragment Recovery



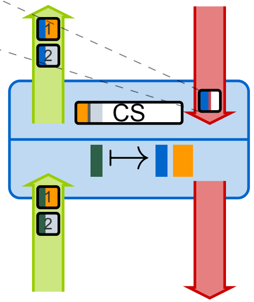
(ACK window size = 2)

Design of the Virtual Reassembling Endpoint

Reassemble tentatively in CS *en-route*



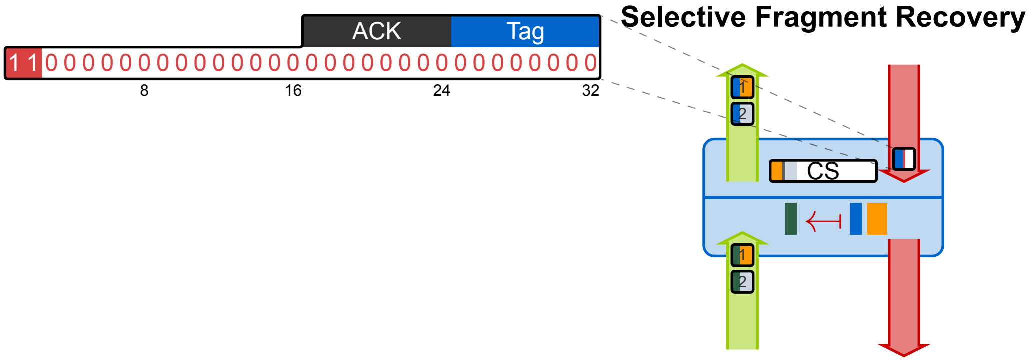
Selective Fragment Recovery



(ACK window size = 2)

Design of the Virtual Reassembling Endpoint

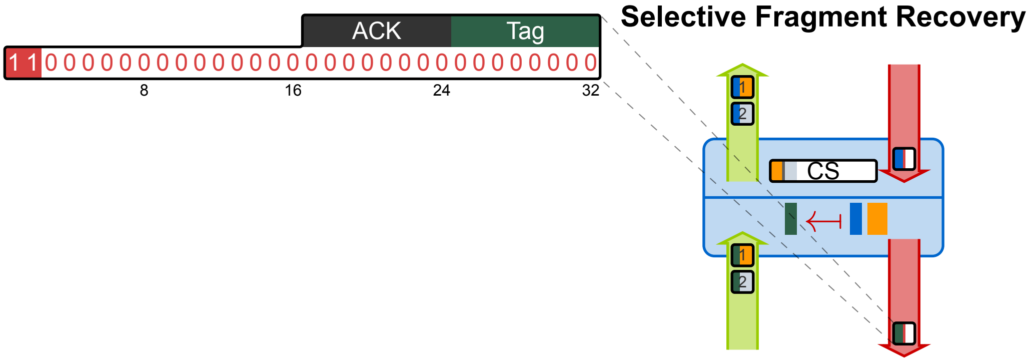
Reassemble tentatively in CS *en-route*



(ACK window size = 2)

Design of the Virtual Reassembling Endpoint

Reassemble tentatively in CS *en-route*

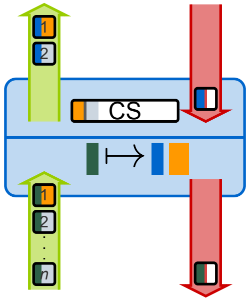


(ACK window size = 2)

Design of the Virtual Reassembling Endpoint

Reassemble tentatively in CS *en-route*

Selective Fragment Recovery

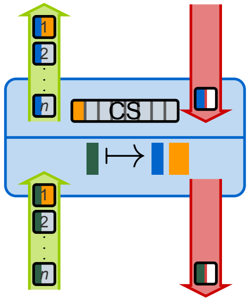


(ACK window size = 2)

Design of the Virtual Reassembling Endpoint

Reassemble tentatively in CS *en-route*

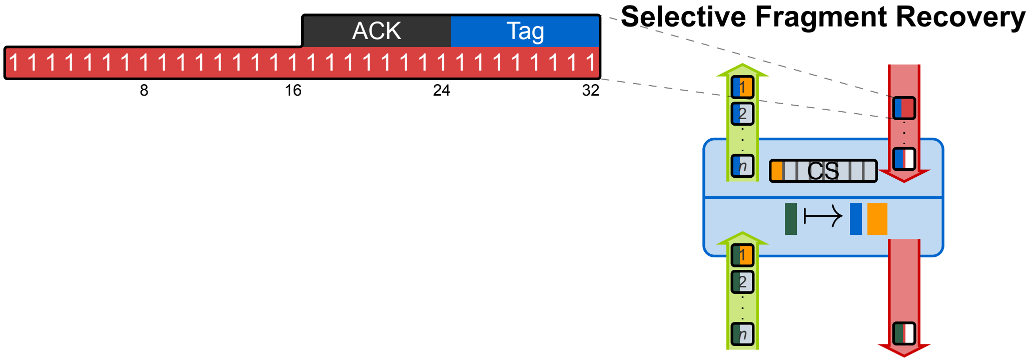
Selective Fragment Recovery



(ACK window size = 2)

Design of the Virtual Reassembling Endpoint

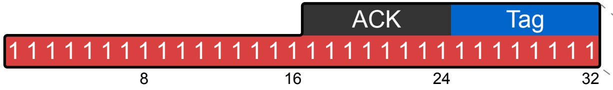
Reassemble tentatively in CS *en-route*



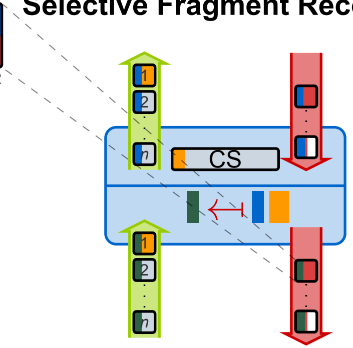
(ACK window size = 2)

Design of the Virtual Reassembling Endpoint

Reassemble tentatively in CS *en-route*



Selective Fragment Recovery



(ACK window size = 2)

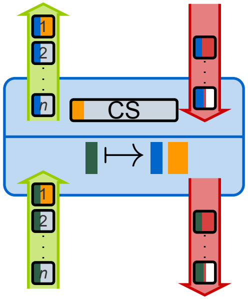
- Full ACK enables CS entry to be served to incoming Interests

Design of the Virtual Reassembling Endpoint

Reassemble tentatively in CS *en-route*

Another benefit:

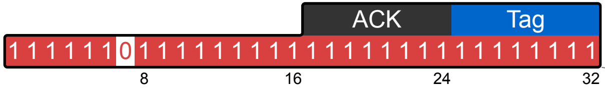
Selective Fragment Recovery



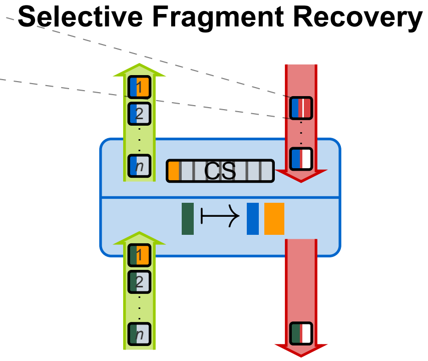
(ACK window size = 2)

Design of the Virtual Reassembling Endpoint

Reassemble tentatively in CS *en-route*



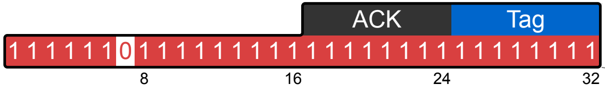
Another benefit: Restore lost fragments



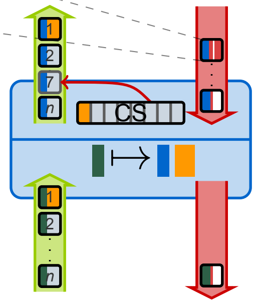
(ACK window size = 2)

Design of the Virtual Reassembling Endpoint

Reassemble tentatively in CS *en-route*



Selective Fragment Recovery

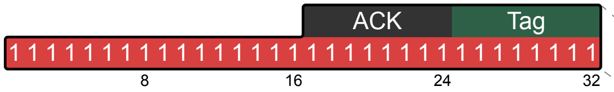


(ACK window size = 2)

Another benefit: Restore lost fragments

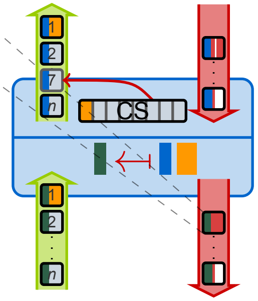
Design of the Virtual Reassembling Endpoint

Reassemble tentatively in CS *en-route*



Another benefit: Restore lost fragments

Selective Fragment Recovery



(ACK window size = 2)

Outline

Motivation & Background

A Virtual Reassembling Endpoint in ICN

Evaluation of the VREP extension

Conclusion & Outlook

Evaluation setup

Comparing 3 fragment forwarding approaches

Hop-wise Reassembly
(*HWR*)

Classic
Selective Fragmente Recovery
(*SFR w/o VREP*)

Selective Fragmente Recovery
with VREP extension
(*SFR w/ VREP*)

Evaluation setup

Comparing 3 fragment forwarding approaches

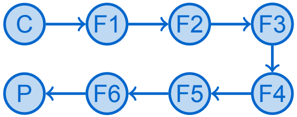
Hop-wise Reassembly
(*HWR*)

Classic
Selective Fragmente Recovery
(*SFR w/o VREP*)

Selective Fragmente Recovery
with VREP extension
(*SFR w/ VREP*)

Deployment at FIT/IoT-LAB testbed in Grenoble:

Daisy chain topology
(benefits caching)


$$\begin{tikzcd} C \arrow[r] & F1 \arrow[r] & F2 \arrow[r] & F3 \arrow[d] \\ P & F6 \arrow[l] & F5 \arrow[l] & F4 \arrow[l] \end{tikzcd}$$

Evaluation setup

Comparing 3 fragment forwarding approaches

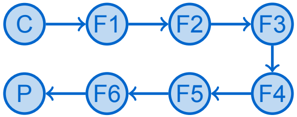
Hop-wise Reassembly
(*HWR*)

Classic
Selective Fragmente Recovery
(*SFR w/o VREP*)

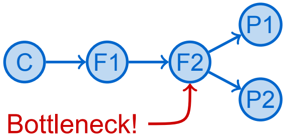
Selective Fragmente Recovery
with VREP extension
(*SFR w/ VREP*)

Deployment at FIT/IoT-LAB testbed in Grenoble:

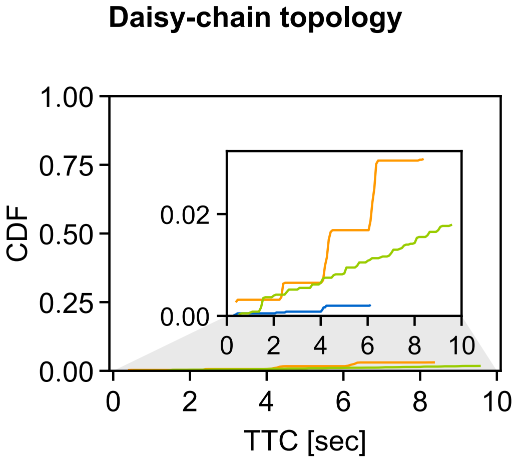
Daisy chain topology
(benefits caching)

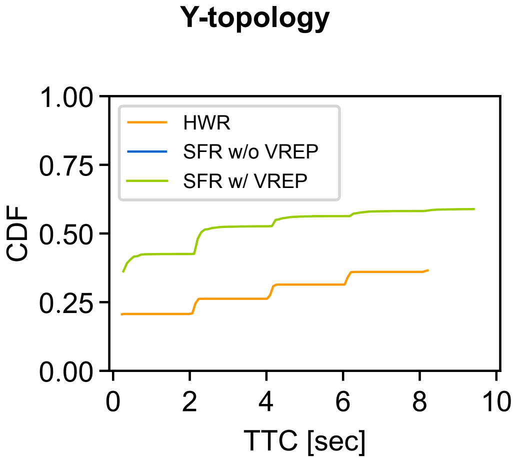


Y-topology
(benefits fragment forwarding)

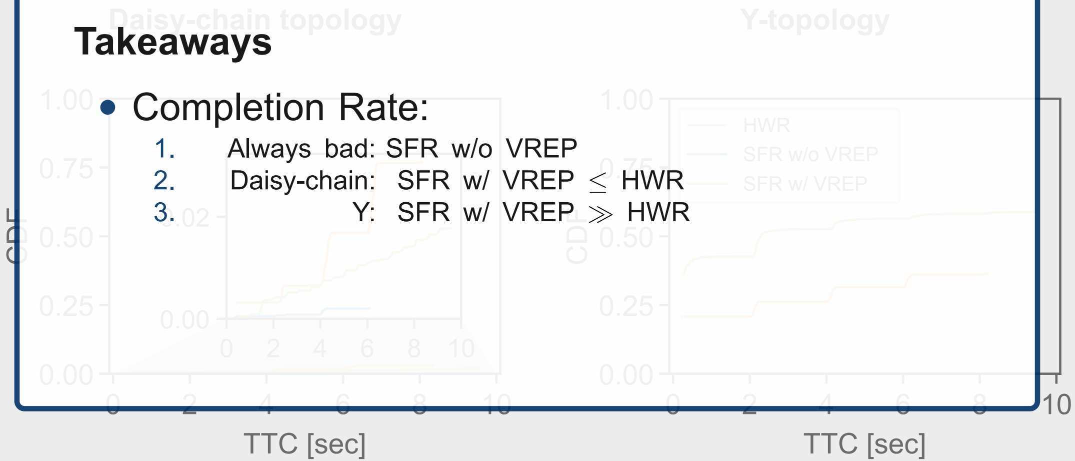


Time to Completion (TTC)





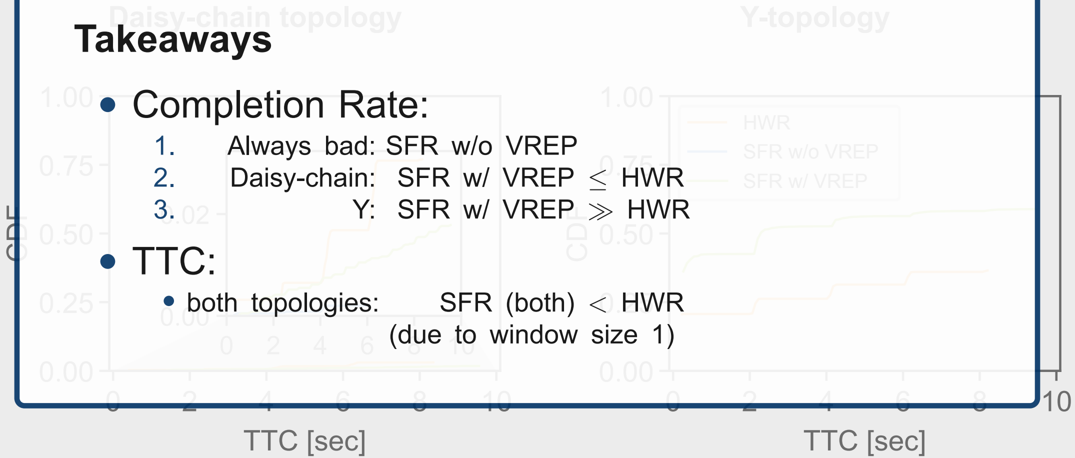
Time to Completion (TTC)



Takeaways

- Completion Rate:
  1. Always bad: SFR w/o VREP
  2. Daisy-chain: SFR w/ VREP $\leq$ HWR
  3. Y: SFR w/ VREP $\gg$ HWR

Time to Completion (TTC)

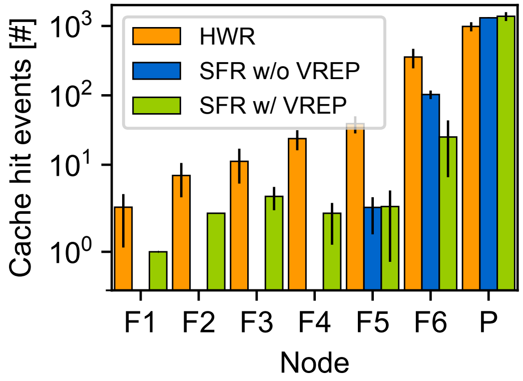


Takeaways

- Completion Rate:
  1. Always bad: SFR w/o VREP
  2. Daisy-chain: SFR w/ VREP $\leq$ HWR
  3. Y: SFR w/ VREP $\gg$ HWR
- TTC:
  - both topologies: SFR (both) $<$ HWR (due to window size 1)

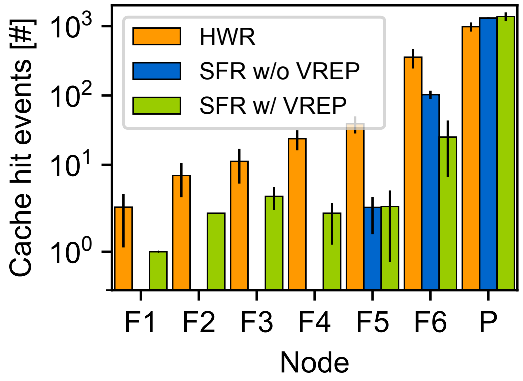
Cache (CS) Hits

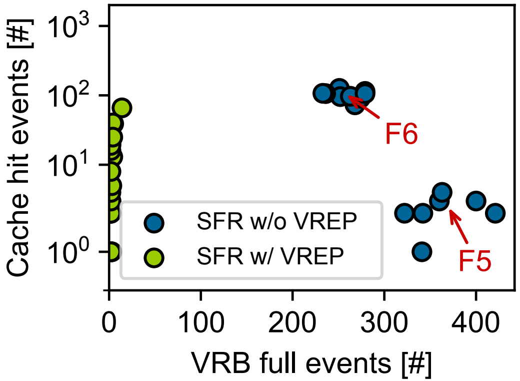
Daisy-chain topology



Cache (CS) Hits

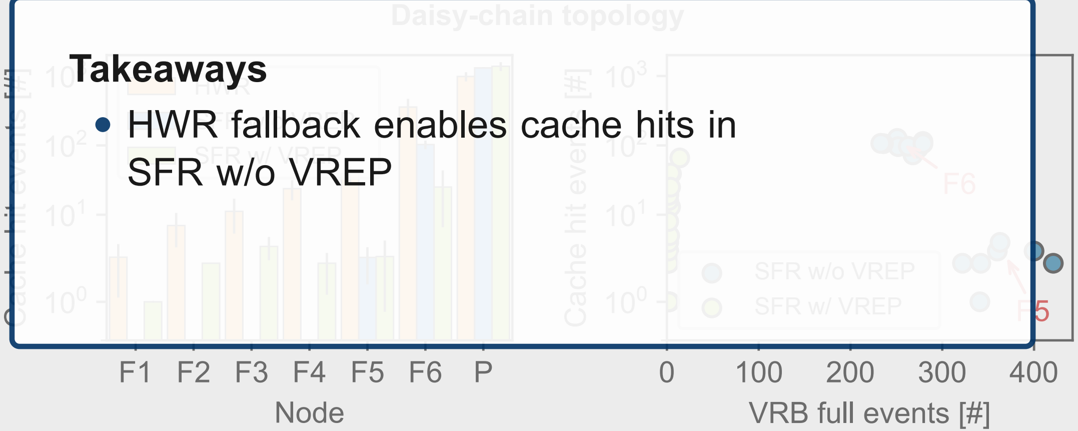
Daisy-chain topology





(SFR in RIOT falls back to HWR when Virtual Reassembly Buffer [VRB] is full)

Cache (CS) Hits

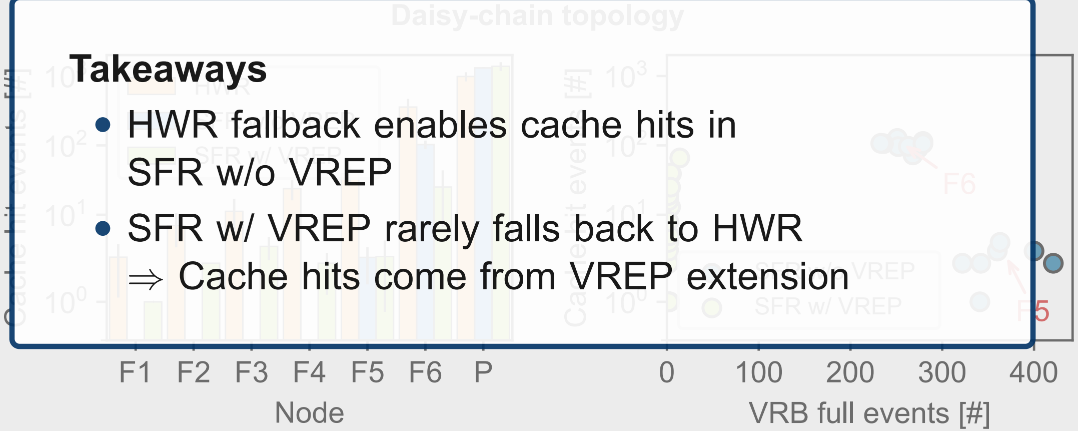


Takeaways

- HWR fallback enables cache hits in SFR w/o VREP

(SFR in RIOT falls back to HWR when Virtual Reassembly Buffer [VRB] is full)

Cache (CS) Hits



Takeaways

- HWR fallback enables cache hits in SFR w/o VREP
- SFR w/ VREP rarely falls back to HWR
  $\Rightarrow$ Cache hits come from VREP extension

(SFR in RIOT falls back to HWR when Virtual Reassembly Buffer [VRB] is full)

Outline

Motivation & Background

A Virtual Reassembling Endpoint in ICN

Evaluation of the VREP extension

Conclusion & Outlook

Conclusion

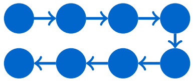
- Virtual Reassembling Endpoint (VREP) extension allows for
  - Minimal memory usage
  - Full content chunk being served to ICN forwarders

Conclusion

- Virtual Reassembling Endpoint (VREP) extension allows for
    - Minimal memory usage
    - Full content chunk being served to ICN forwarders
- In any case bad: Selective Fragment Recovery (SFR) w/o VREP

Conclusion

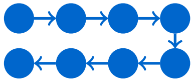
- Virtual Reassembling Endpoint (VREP) extension allows for
  - Minimal memory usage
  - Full content chunk being served to ICN forwarders
- In any case bad: Selective Fragment Recovery (SFR) w/o VREP
- Choice between Hop-wise Reassembly (HWR) and SFR w/ VREP depends on topology:



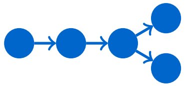
HWR $\geq$ SFR w/ VREP

Conclusion

- Virtual Reassembling Endpoint (VREP) extension allows for
  - Minimal memory usage
  - Full content chunk being served to ICN forwarders
- In any case bad: Selective Fragment Recovery (SFR) w/o VREP
- Choice between Hop-wise Reassembly (HWR) and SFR w/ VREP depends on topology:



HWR $\geq$ SFR w/ VREP



HWR $\ll$ SFR w/ VREP

Next steps

Evaluation of Virtual Reassembling Endpoint extension impact with

- More complex MAC layers, e.g. IEEE 802.15.4e
- Congestion control mechanisms within SFR
- Different ICN caching strategies

Reproducible research? Yes!
Find the code of our experiments at

https://github.com/5G-I3/ACM-ICN-2020-SFR
or in the ACM Digital Library.

m.lenders@fu-berlin.de

ICNLoWPAN: ICN adaption to LoWPAN networks

Challenge

- IEEE 802.15.4: 127 bytes PDU

Common approaches

- Header compression + **Fragmentation**

ICNLoWPAN also supports *PLC*, BLE, etc.

ICNLoWPAN: ICN adaption to LoWPAN networks

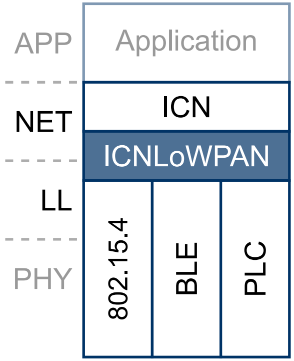
Challenge

- IEEE 802.15.4: 127 bytes PDU

Common approaches

- Header compression + **Fragmentation**

ICNLoWPAN also supports *PLC*, BLE, etc.



Memory requirements in reassembly buffer (RB)

(L2 src, DG tag)

Memory requirements in reassembly buffer (RB)

(L2 src, DG tag)

Link layer address (incl. length indicator) $\leq$ **8 bytes** $+$ **1 byte**

Memory requirements in reassembly buffer (RB)

(L2 src, DG tag)

Link layer address (incl. length indicator)	$\leq$ **8 bytes** $+$ **1 byte**
Datagram tag	$=$ **2 bytes**

Memory requirements in reassembly buffer (RB)

(L2 src, DG tag)

Link layer address (incl. length indicator)	$\leq$ **8 bytes** $+$ **1 byte**
Datagram tag	$=$ **2 bytes**
Datagram itself	$\leq$ **1280 bytes**
Per RB entry	**1291 bytes** $\approx$ **1.25 KiB**

Memory requirements in reassembly buffer (RB)

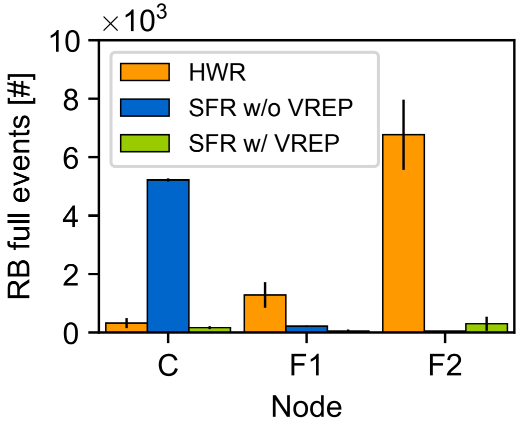
(L2 src, DG tag)

Link layer address (incl. length indicator)	$\leq$ **8 bytes** $+$ **1 byte**
Datagram tag	$=$ **2 bytes**
Datagram itself	$\leq$ **1280 bytes**
Per RB entry	**1291 bytes** $\approx$ **1.25 KiB**
Typical IoT node RAM	$\sim$ **10 KiB** $-$ **50 KiB**

$\Rightarrow$ Many implementations: only 1 reassembly buffer entry.

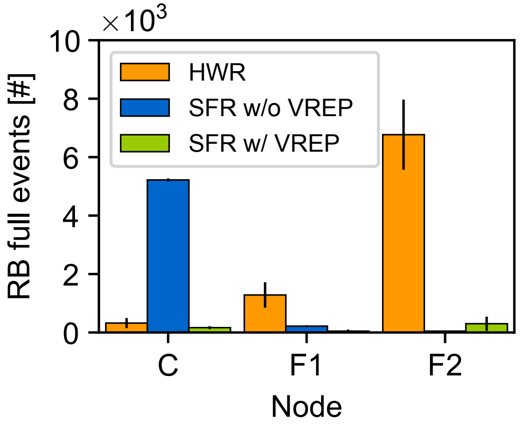
Reassembly Buffer (RB) usage

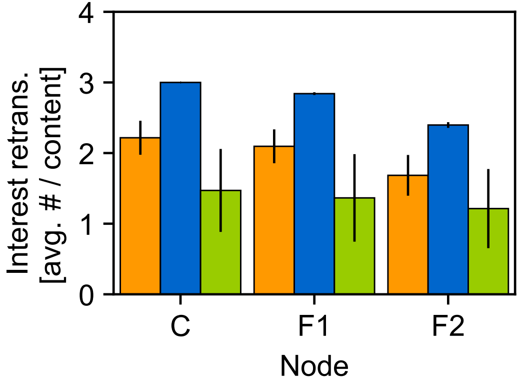
Y-topology



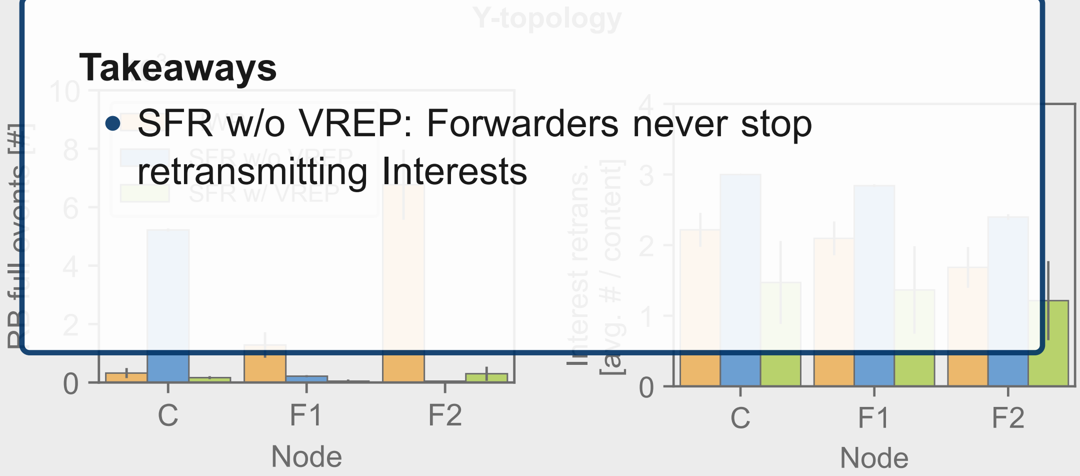
Reassembly Buffer (RB) usage

Y-topology





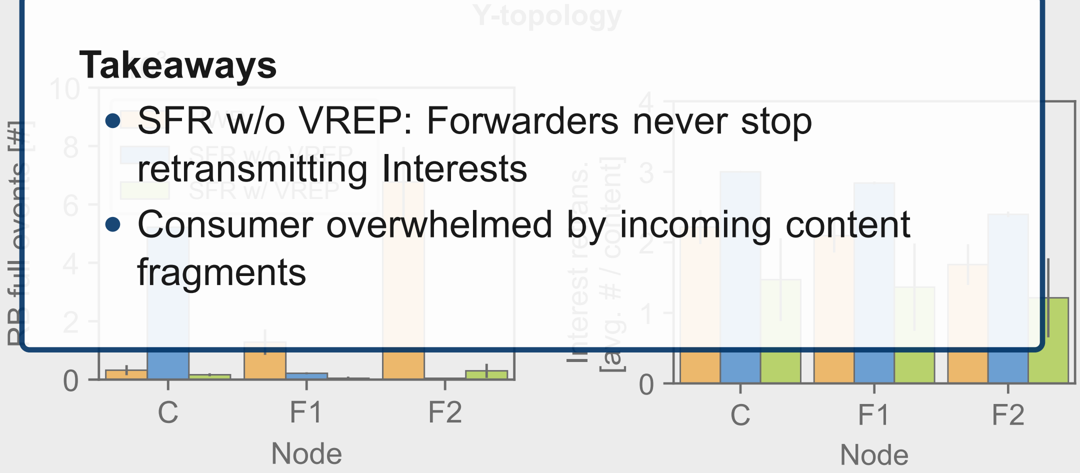
Reassembly Buffer (RB) usage



Takeaways

- SFR w/o VREP: Forwarders never stop retransmitting Interests

Reassembly Buffer (RB) usage



Takeaways

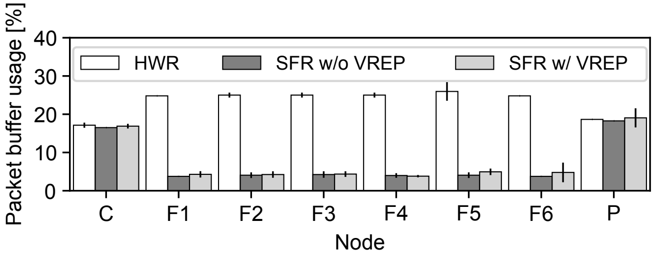
- SFR w/o VREP: Forwarders never stop retransmitting Interests
- Consumer overwhelmed by incoming content fragments

Memory Consumption

	HWR	SFR w/o VREP	SFR w/ VREP
ROM	1.7 KiB	5.5 KiB	6.4 KiB
RAM	1.2 KiB	5.2 KiB	7.5 KiB

Memory Consumption

	HWR	SFR w/o VREP	SFR w/ VREP
ROM	1.7 KiB	5.5 KiB	6.4 KiB
RAM	1.2 KiB	5.2 KiB	7.5 KiB



(Packet buffer size: 8 KiB)