Urgency Based Scheduler

Scalability - How many Queues?!

Johannes Specht johannes.specht AT uni-due.de **Univ. of Duisburg-Essen**

Soheil Samii soheil.samii AT gm.com **General Motors**

Contents

UBS

- ... provides low latency guarantees with high link utilization even without scheduling
- … allows mapping of flow latency requirements
- … supports different types of flow traffic patterns
- … has integrated safety properties

UBS

- … has network dependencies (number of queues)
- … which is challanging to predict for implementers to satisfy target market needs (scaling)

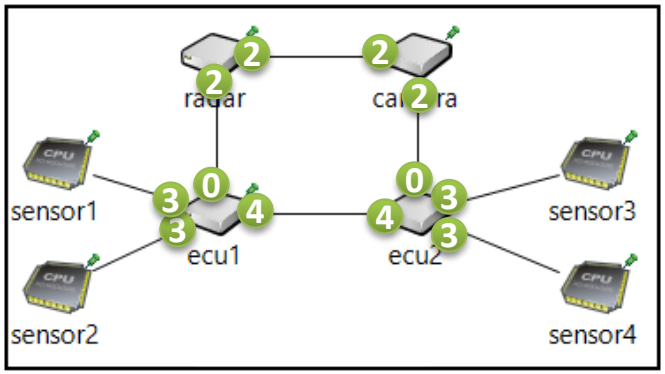
Goals of this slide set

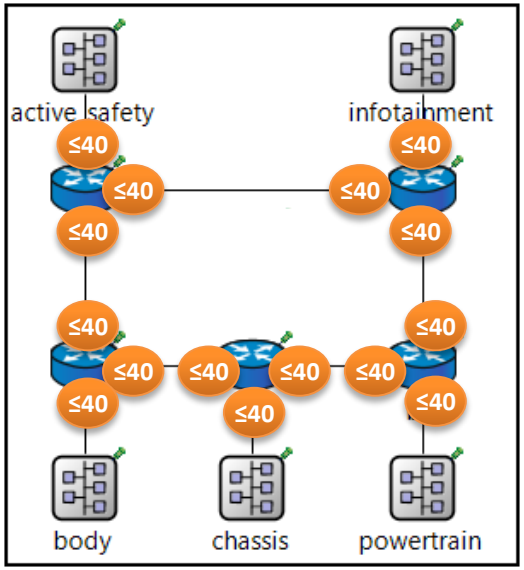
- Discuss the network dependencies
- Consider UBS from an implementers point of view (based on feedback and experiments with FPGAs)
- Determine limits for the number of queues
- Discuss a scalable UBS derivate

No Goals of this slide set

- Looking for ultimately low latency QoS algorithms – this is 802.1Qbv + 802.1Qbr …
  - it gives the ultimate low latency
- … to the price of:
  - Network-wide planning and cycle synchronization, time scheduling
  - Network synchronous periodic talker transmissions
  - …

Network Dependencies …





ADAS Example

The number of flows/queues for an ADAS system in isolation

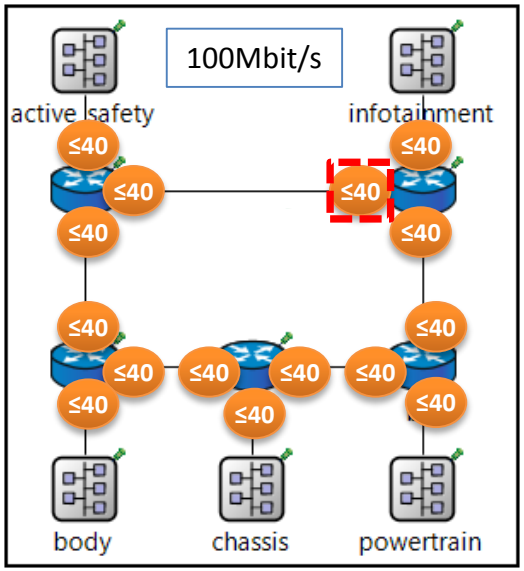
- *Relaxed*
- there is just a had full of flows per egress port. One shaped queue per flow is not so much…

Backbone Example

.. But *the same bridge* may be used as e.g. a automotive backbone bridge

- *This becomes "inconvenient"…*
- there could be more flows at an egress port, in the worst case each one in a separate shaped queue – how much queues are needed?!

Network Dependencies … Part 2



UBS would do a really good job here…

Assume the following example setup at the red marked port:

- **3 x video**
  - 20 Mbit/s, 1542 B. max. packet
- **5 x perioric status information**
  - 40 ms period, 100 B. max. packet
  - → 20 kbit/s
- **5 x traditional periodic control**
  - 5 ms period, 84 B. max. packet
  - → 134.4 kbit/s
- **5 x large packet control (e.g. radar)**
  - 5 ms period, 625 B. max. packet
  - → 1 Mbit/s

Not that time critical

- Put in a low sub-priority at this port
- Experience higher latency at this hop, give room for high sub priority flows

Harder latency requirements

- Put in a high sub-priority at this port
- Experience lower latency at this hop, use the room given by the other flows

At other ports:

- A totally different constellation of flows may exist…
- … with other requirements and/or
- Sub priorities may be assigned differently to …
- finally map the End to End latency requirements of all flows

UBS per Flow Dependencies Analyzed

Sub-Shaper State per Flow

Assumed to be **no issue**:

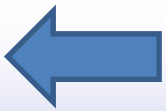
- Each sub-shaper requires bucket state, like all bucket based algorithms
- Per flow ingress policing for rate limited flows requires bucket state as well (maybe even more if dual-buckets are used)
- At egress, UBS provides similar protection like per flow policing (802.1Qci) at ingress

→ In both cases, roughly equivalent per flow bucket state (e.g. SRAM) is needed. It's just the direction of the arrow, which is different (ingress/egress).

Sub-Queues per Flow

Assumed to be **a real issue**:

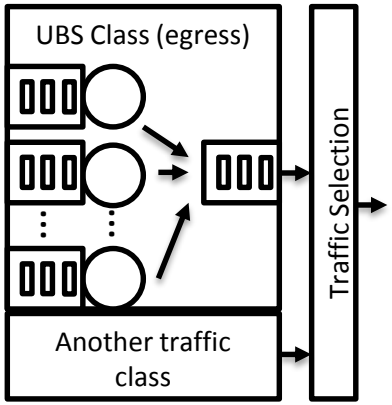
- Queues(packet buffers and linked lists) = per queue state + packet buffer capacity
  - **Per queue state** (at least head and tail pointers) grows with the number of queues, i.e. flows
    → Issue, but assumed to be **a „not that hard" issue (see bucket state equivalence)**
  - **Per queue buffer** is a function of the maximum per hop latency - per hop latency is proportional to the overall buffer capacity
    → Assumed to be **no issue (UBS is deterministic and gives reasonable low latency)**
  - **Queue transmission selection order** depends on sub-shapers, which can *"fire" packets in dynamic order* (unlike e.g. static round-robin or TAS).
    → Assumed to be **the real issue**



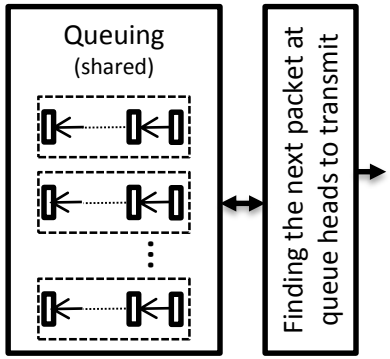
Again, a closer look on the next slides

Queues

Logically, as shown in IEEE



Slightly closer to implementations



Finding the next packet to transmit

- Once per min. packet duration (~84 Byte times), a packet for transmission must be found, i.e. the next head element of the linked lists.
- If there are only a few queues (like in the ADAS example) and link speed is low (100Mb/s or 1Gb/s), the right queue may be selected even in software by iterating over the queues. If there are much more queues…
- … dynamically changing order between the queues either requires parallel logic for each on each comparisons (more area), update an ordered data structure on each packet transmission or a combination of both
- Ordered data structure examples:
  - *Heaps(Tree)* → *O(log N)* → ***Small***, but ***slow*** but for lots of elements (i.e. flows)
  - *Calendars* → *O(1)* → ***Fast*** but ***large*** (depending on its time-range and resolution)

Notes:
Compared to the logical picture, it is assumed that implementations would not physically implement the logical common queue before traffic selection but pick the next packet to transmit directly from the head elements of the per flow queues to the left. Consecutive slides will not account the rightmost FIFO queues in the logical picture.

A SCALABLE UBS DERIVATE

Overview

Difference: Shared Queues – One FIFO queue for many flows from an ingress port

- **Assumption:**
  The number of ports of a bridge is known at design time.
- **Rule Set:**
  One egress queue is shared by *all* flows from the same ingress port if:
  - The neighbor at the ingress ports send's the flows in the same sub-priority
  - The egress queue at the local bridge sends the flows in the same sub-priority

  → No per flow dependencies
- **Queue operation:**
  FIFO → no reordering or similar
- Transmission decision based on queue heads → no look-ahead deep into queues, etc.

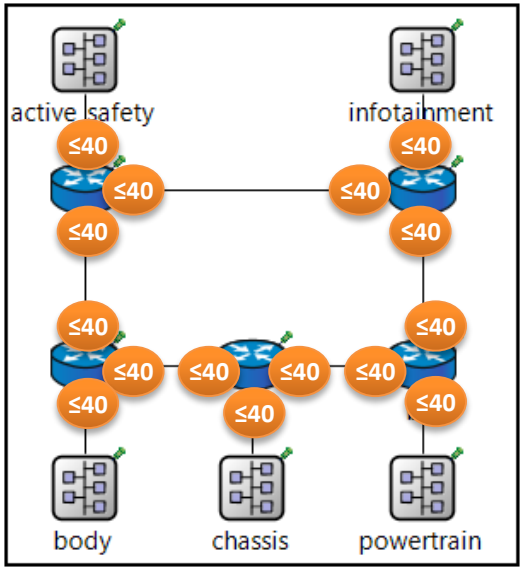
What does this mean for implementations supporting UBS?

- A bridge ***has to*** provide (port count – 1) queues at egress for UBS.
- A bridge ***may*** provide more queues for UBS, thus allowing users to use sub-priorities, fine grained flow isolation schemes, etc...
- However, this is the decision is up the implementer and seems to be nothing new, i.e. in AVB gen1 the decision about the number shaped traffic classes is also up to the implementer

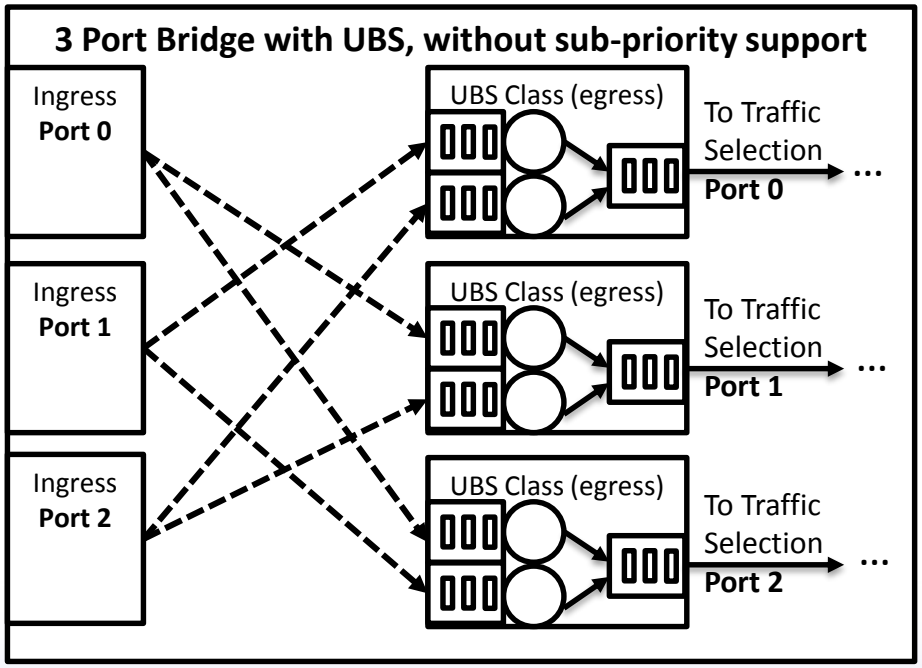
Back to the Backbone Example …

Without sub-priorities (mandatory part)

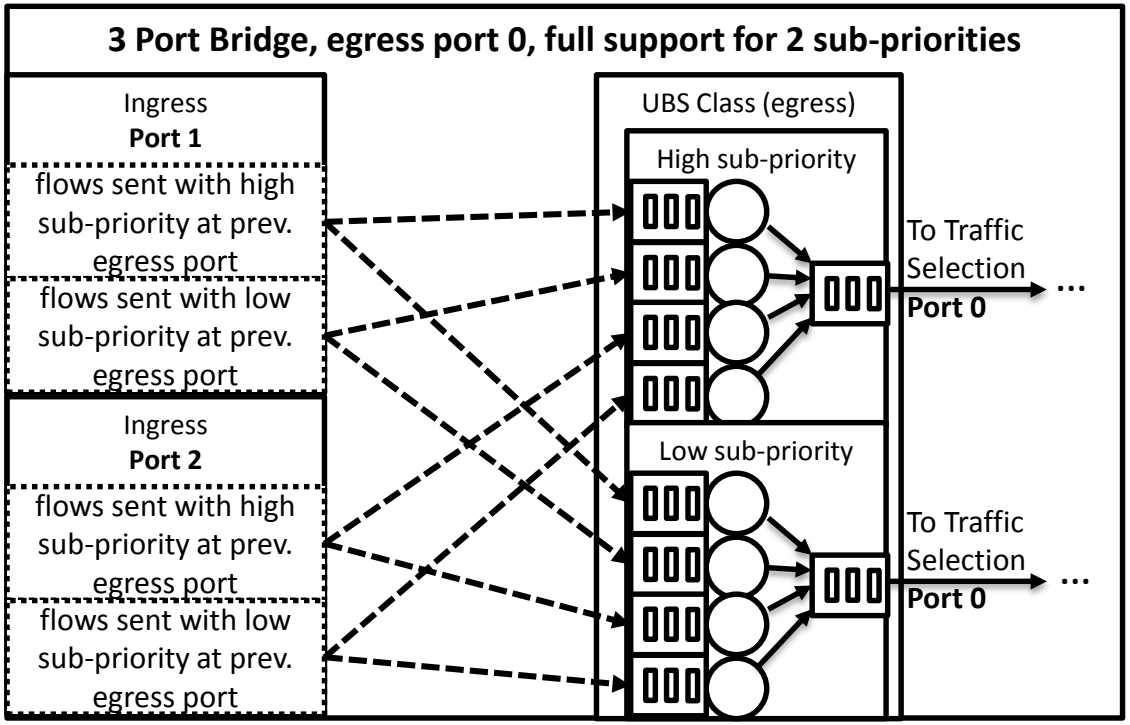
- Each of the 5 bridges has to provide (ingress ports-1) queues at egress to join UBS communication.
- I.e.: *2 FIFO Queues* per egress port in a *3-port bridge* can transport 40, 1000 (or more) UBS flows.



3 Port Bridge, logically: <u>All queues</u> at egress



… With Sub-Priorities …

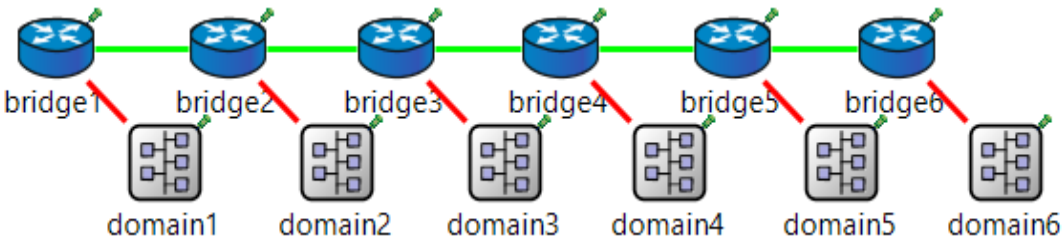


Supporting *arbitrary per flow* sub-priorities at any port in the network

Multiplier is *n times* $n=n^2$ (*n*=sub-priorities)

- First *n* to allow higher sub-priority packets to overtake lower sub-priority packets at egress
- Second *n* is *implementation dependent*:
  Assumes implementations would like to serve queues in a strict FIFO manner at a finite rate, avoids that packets for high priority transmissions are hidden behind multiple packets for low priority transmission*

Somewhere in the Middle



Green links:
- Sub-priorities

Red links:
- No sub-priorities

Arbitrary combinations may be rare

Dropping the multiplier from n^2 to n seems possible in automotive and industrial networks:

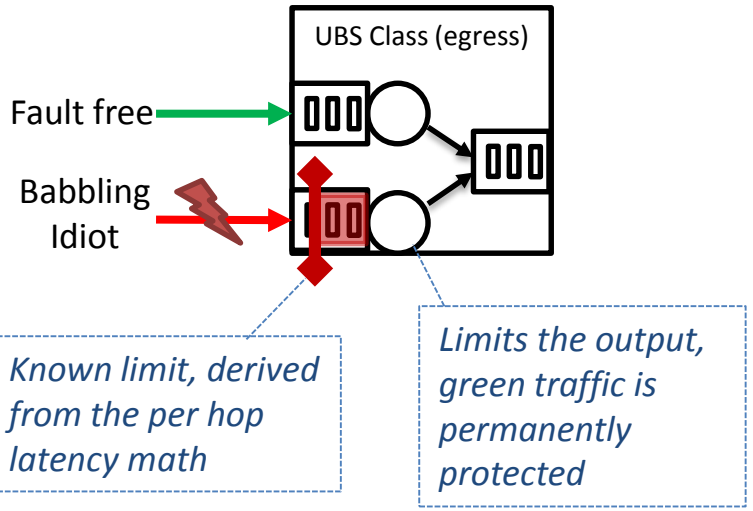
- UBS brings speed to low latency flows by juggling with priorities at long linear paths
- The link from/to the path can work without sub-priorities in both directions (from and to the ring or chain)

→ 4 queues per port for a 3 port bridge supporting 2 sub-priorities

UBS Protection vs. Shared Queues

Recap: UBS, as presented earlier, had two essential safety properties

1. No dependency on Clock Synchronization:
   In a safety critical system, clock sync. is a single point of failure, if communication depends on it. If clock sync fails, transport of critical flows is no longer assured.
2. Isolation of flows:
   - Shaped per flow queue assured that babbling idiots don't interfere with fault free flows.
   - Further actions (flow blocking, port blocking) could be triggered by exceeding known queue limits.
   - Exact fault isolation was possible.



Impact of shared Queues

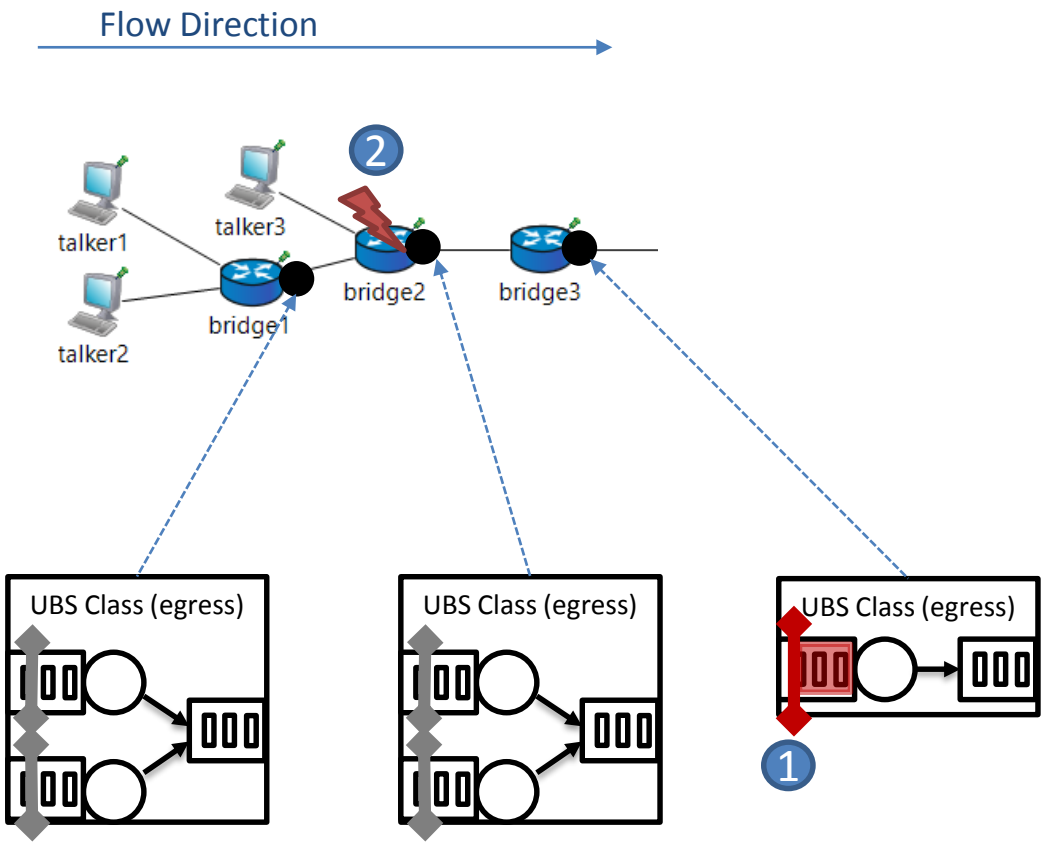
Obvious:

- Still no dependency on clock sync.
- Queue limits can still be calculated and set
- 100% protection is still assured

Less obvious:

- Exact fault isolation is still possible, as long as the following fault assumption holds:
  1. *One box fails at a time*
  2. *If "one box" fails, the box fails entirely, affecting all flows from this box, i.e. All flows from this box are considered faulty*

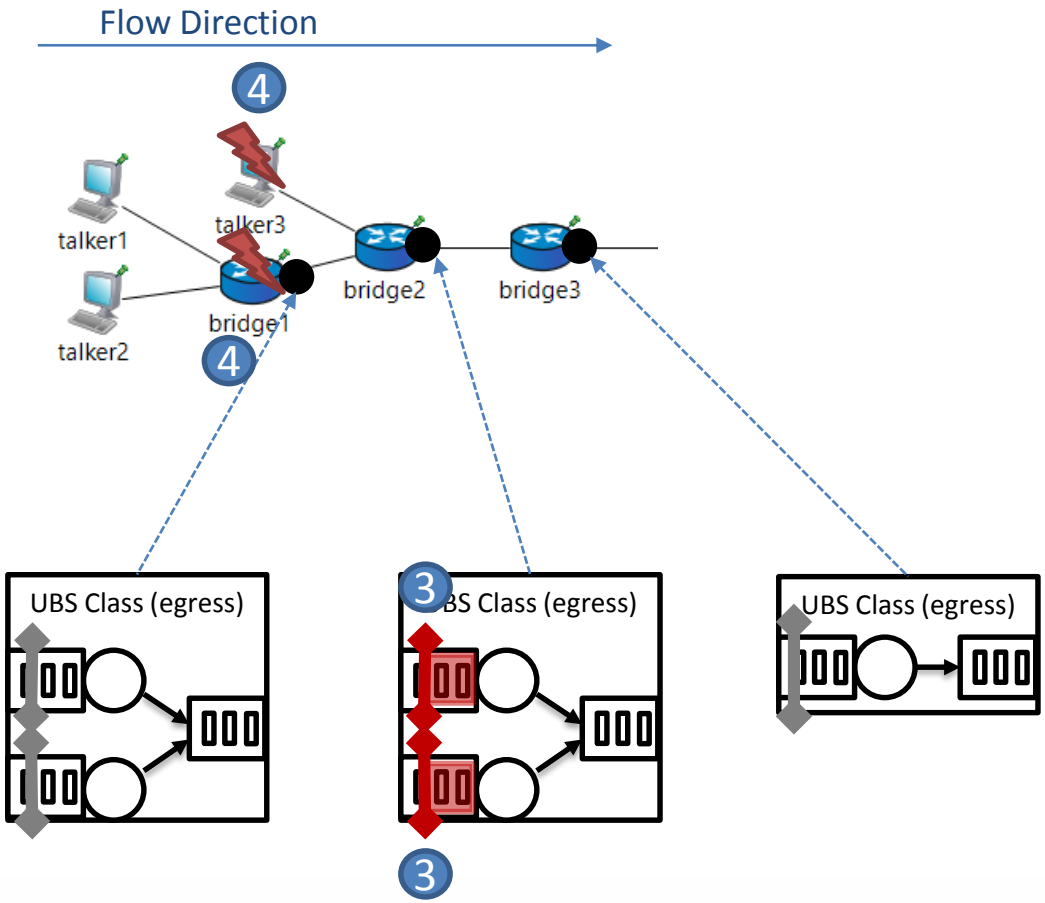
Fault Isolation by Queue Limits (1)



Fault isolation Logic

1. If the queue limit in *bridge3* is exceeded…
2. … only *bridge2* can be the babbling idiot.

Fault Isolation by Queue Limits (2)



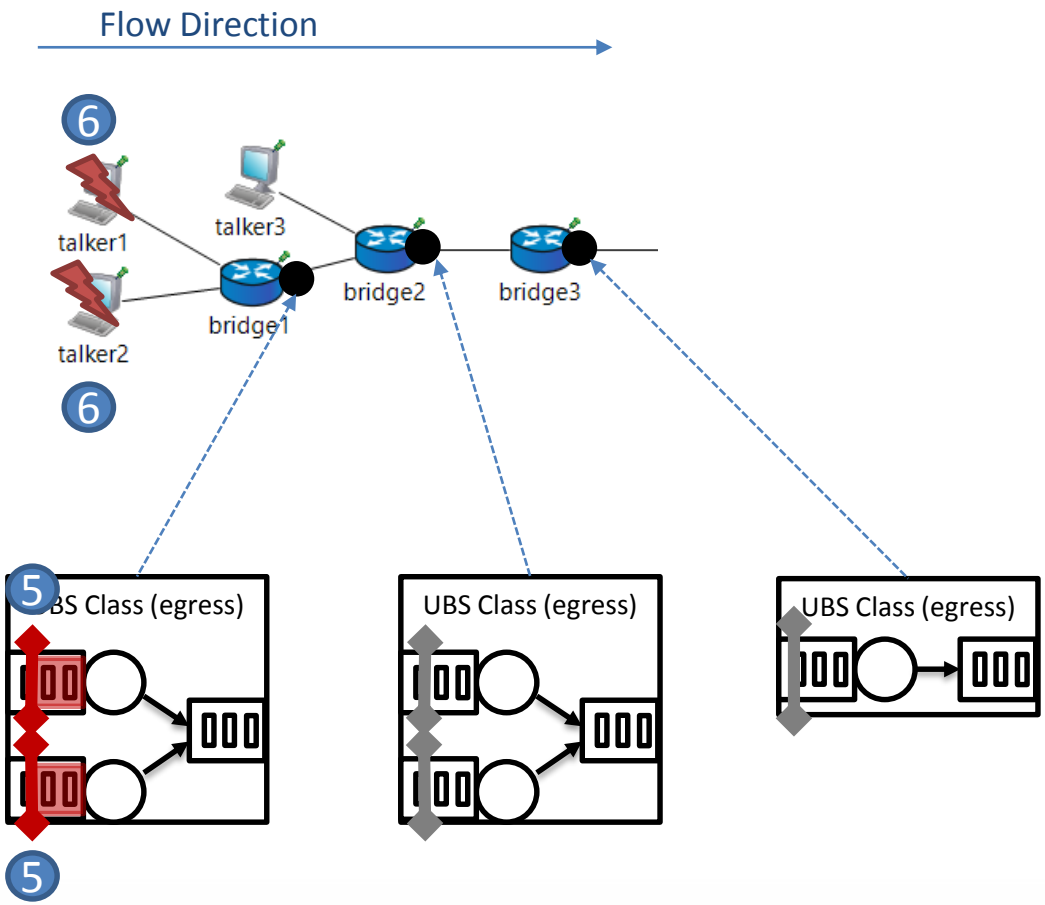
Fault isolation Logic

1. If the queue limit in *bridge3* is exceeded…
2. … only *bridge2* can be the babbling idiot.

Contradiction

3. If the queue limit in *bridge3* is exceeded and *bridge1* or *talker3* would be the babbling idiot…
4. … limits in *bridge2* would prevent the overload to propagate to *bridge3*.

Fault Isolation by Queue Limits (3)



Fault isolation Logic

1. If the queue limit in *bridge3* is exceeded…
2. … only *bridge2* can be the babbling idiot.

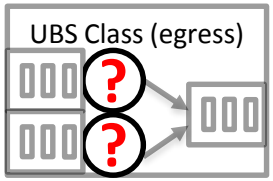
Contradiction

3. If the queue limit in *bridge3* is exceeded and *bridge1* or *talker3* would be the babbling idiot…
4. … queue limits in *bridge2* would prevent the overload to propagate to *bridge3*.

… Continuing …

5. If a queue limit in *bridge2* is exceeded, *bridge1* would be fault free and *talker1* or *talker2* would be the babbling idiot…
6. … queue limits in *bridge1* would prevent the overload to propagate to *bridge2*.

Picking – Algorithm (1)



Serving the queue number *q*:

```
while (true){
        wait until queue[q].containsFrame();

        pkt = queue[q].removeHead();
        f = perFlowStateIndexOf(pkt);

        if (pkt.bitLength > perFlowState[f].maxPktBitLength){
                return -1;

        t = max(now, perFlowState[f].nextDeqeueTime);
        wait until t;
        commonQueue.insert(pkt);
        perFlowState[f].nextDequeueTime =
                t + pkt.bitSize*linkSpeed/perFlowState[f].bitrate;
}
```

- *There is still per flow state - assumed to even out with the built-in protection (see prev. slides)*
- *Index lookup seems to be a O(log n) operation, BUT it is not: Could be done at FDB lookup, by "clever" index allocation, etc.*

Unexpected large packet would congest the queue on the right → Stop operation

Reserved bandwidth of flow f (constant)

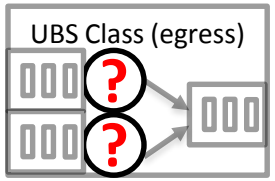
Picking – Algorithm (2)

```
while (true){
          wait until queue[q].containsFrame();

          pkt = queue[q].removeHead();
          f = perFlowStateIndexOf(pkt);

          if (pkt.bitLength > perFlowState[f].maxPktBitLength){
                    return -1;

          t = max(now, perFlowState[f].nextDeqeueTime);
          wait until t;
          commonQueue.insert(pkt);
          perFlowState[f].nextDequeueTime =
                    t + pkt.bitSize*linkSpeed/perFlowState[f].bitrate;
}
```



Model vs. Implementation

- This algorithm is a model, running the loop at inifinite speed (i.e., not tied to link rate)
- Implementation would (should?) be different but showing the behavior
  Example:
  1. iterate over queue heads …
  2. … find the next head element to transmit (based on nextDequeueTime and sub-priority)
  3. Done once per packet transmission (i.e. in worst case once every 84 byte times)

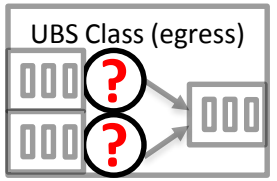
Picking – Algorithm (3)

```
while (true){
          wait until queue[q].containsFrame();

          pkt = queue[q].removeHead();
          f = perFlowStateIndexOf(pkt);

          if (pkt.bitLength > perFlowState[f].maxPktBitLength){
                    return -1;

          t = max(now, perFlowState[f].nextDeqeueTime);
          wait until t;
          commonQueue.insert(pkt);
          perFlowState[f].nextDequeueTime =
                    t + pkt.bitSize*linkSpeed/perFlowState[f].bitrate;
}
```



State Reduction

- Switching to a token bucket based algorithm will enable flows in one shared queue sharing state (network dependent)
- However, the presented picking algorithm:
  - Is simpler/easier to understand and verify
  - Is network independent, configuration is straight forward

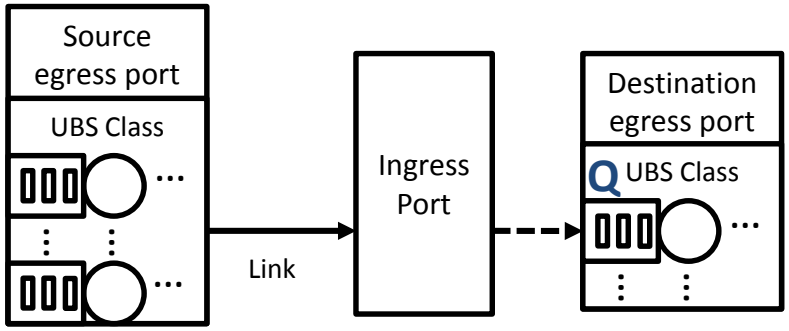
Per Hop Latency Math

$$W_f^{max} \leq \max_{\forall f' \, in \, Q(f)} \left(\frac{\sum_{i \in H} l_i^{max} + \sum_{i \in S} l_i^{max} + \max_{i \in L}(l_i^{max})}{R - \sum_{i \in H} R_i} + \frac{l_{f'}^{max}}{R} \right)$$

Max. over all flows sharing The queue with f in the destination port

Interference by all competing flows In the source egress port (not only The ones in the same queue like f).

S&F



Just the Differences

- For flows in the highest sub-priority level:
  - S&F delay equalizes max over all sharing flows
  - Nothing has changed compared to the math presented in earlier slide sets
- For flows in lower sub-priority levels:
  - May get a slightly higher latency, depending on the packet lengths in this sub-priority level
  - Latency equalized to the max. within this sub-priority level
  - Seems ok, latency requirements of these flows are relaxed anyway

Term	Description
W_f^{max}	Max. per hop delay of a flow f
$\sum_{i \in H} l_i^{max}$	Sum of max. packet lengths of flows with a higher sub-priority than f
$\sum_{i \in S} l_i^{max}$	Sum of max. packet lengths of flows with sub-priority equal to the sub-priority of f
$\max_{i \in L}(l_i^{max})$	Maximum packet length of all flows with a lower sub-pririty than f, including lower priority traffic classes.
l_f^{max}	Maximum packet length of flow f.
R	Link speed.
$\sum_{i \in H} R_i$	Sum. of datarates of flows with a higher sub-priority than f.

Notes:
The shown math does not account minor impacts like oscillator deviations, time discrete operation of devices, etc.

DON‘T. TRUST. MATH.

Benchmarks

Current Status of the Math

- Working on a hard proof …
- Yet a rather conservative/safe upper limit:
  Shared queues are supposed to **bring additional speedup** for flows after packets are in line,
  i.e. after the hop where flows join (needs further analysis)

… yet: Two types of Benchmarks

1. Realistic Benchmarking

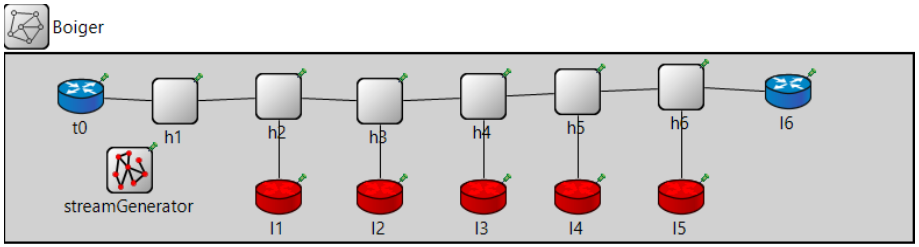
- Fast Ethernet
- No sub-priorities
- Lower priority traffic class interferences (Best Effort)
- 75% link bandwidth reserved at max. for UBS traffic
- Packet lengths limits within Ethernet bounds (84 … 1542 Bytes)

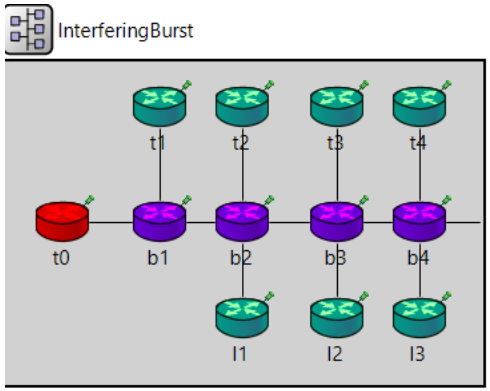
2. Synthetic Benchmarking

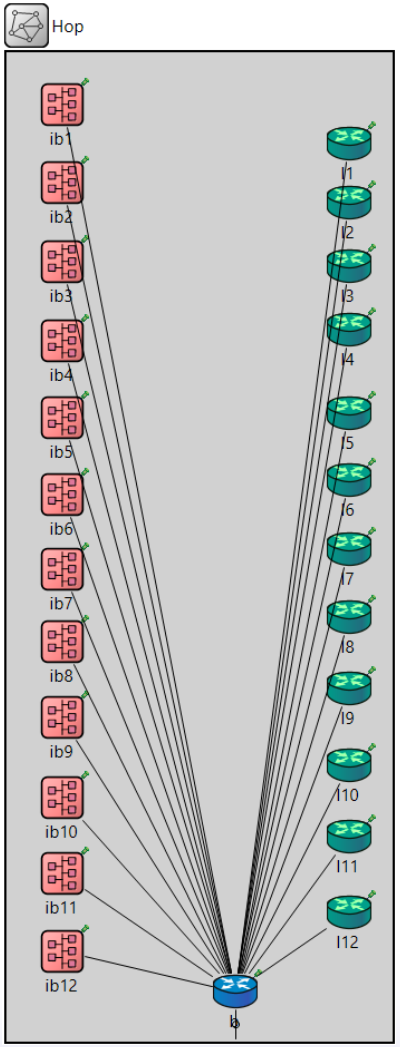
- Gigabit Ethernet (easier to check the math for humans)
- No sub-priorities
- Lower sub priority traffic class interferences (Best Effort)
- ≤ 99% link bandwidth reserved at max. for UBS traffic (scenario dependent)
- Tiny variable packet lengths within:
  - 1..10 bit for UBS flows
  - 1..200 bit for best effort interference

- Construct the „hard cases“ in isolation
- Verify per hop latency math
- Tiny packet length cover more combinatoric space (i.e. “what works with 1..10 bit works also with 672..12336 bit“)

Realistic Benchmark: Christian‘s Case



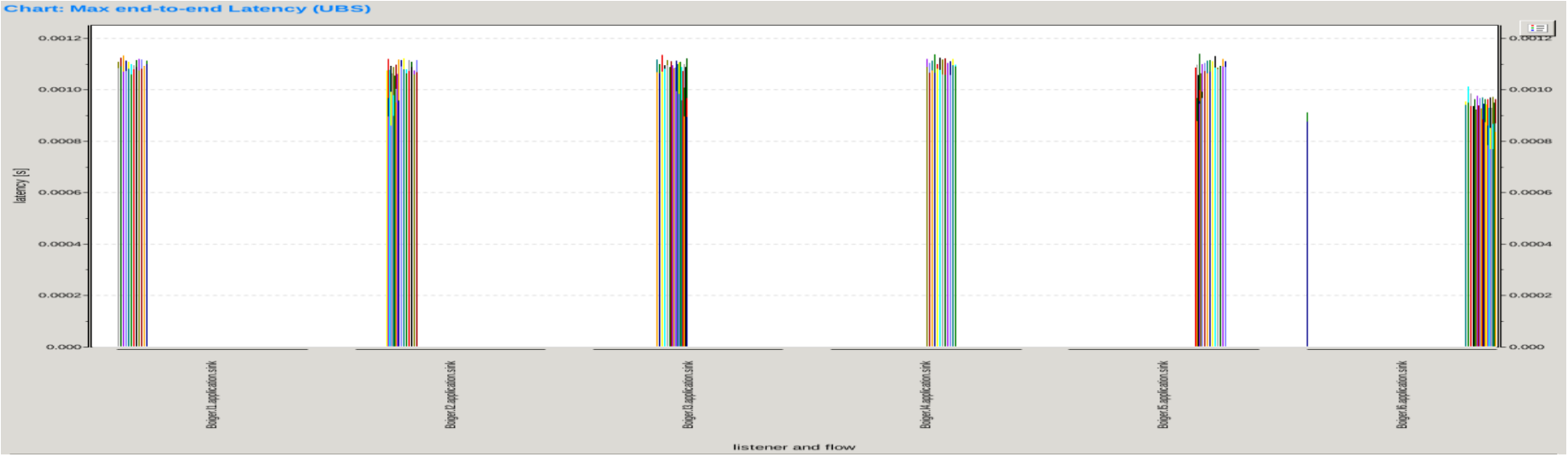




For details flows, see

- http://www.ieee802.org/1/files/public/docs2013/new-tsn-specht-ubs-avb1case-1213-v01.pdf
- http://www.ieee802.org/1/files/public/docs2010/ba-boiger-bridge-latency-calculations.pdf

Results: E2E-Latency

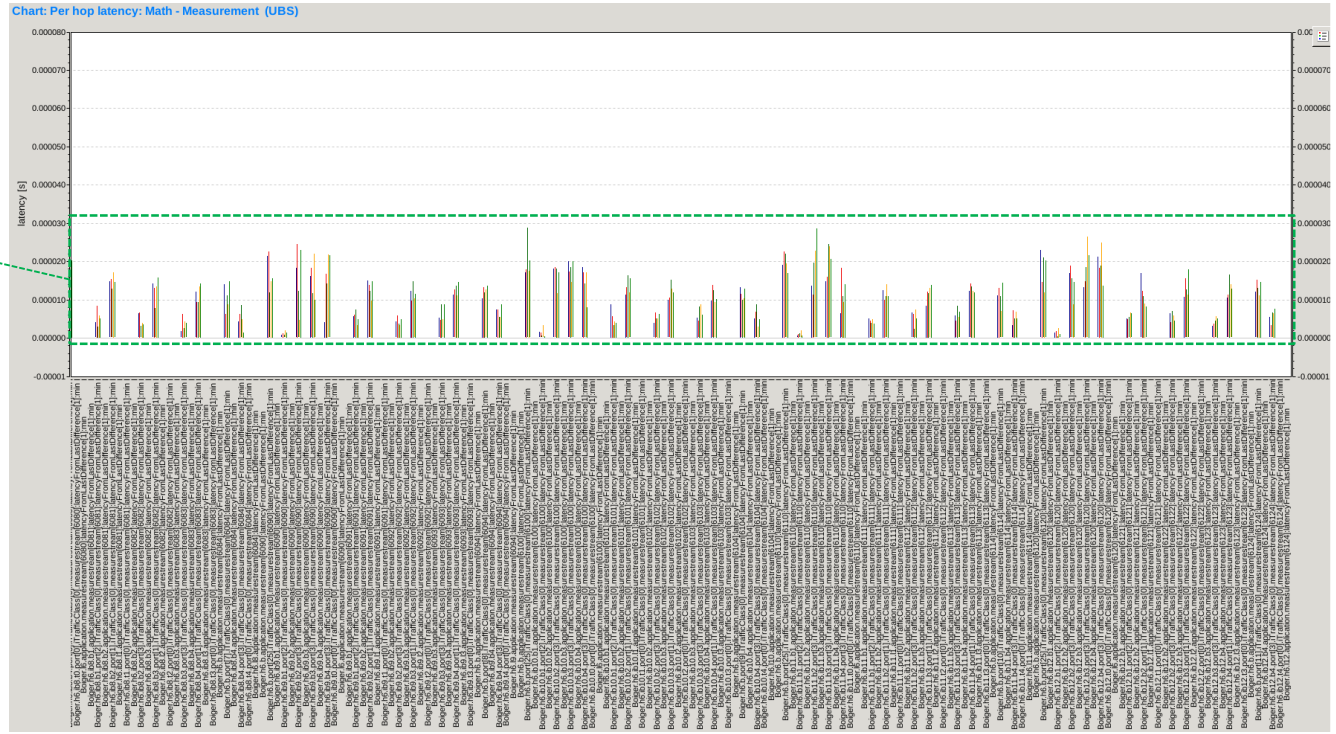


Description

- Measured E2E latency by UBS with shared queues
- At ~70% of the analytical E2E latency (see http://www.ieee802.org/1/files/public/docs2013/new-tsn-specht-ubs-avb1case-1213-v01.pdf)
- However, the primary focus of the benchmarks is on the per hop latency anyway:
  - E2E Latency is just a sum of the per hop latency
  - It‘s unlikely to see a real bad combination over multiple hops purely by looking at the E2E latency
  - → **Better look at the per hop latency …**

Results: Per hop Latency, Math vs. Measurement (proposed UBS Algorithm)

All measurements > 0, i.e. no violations of the latency math.

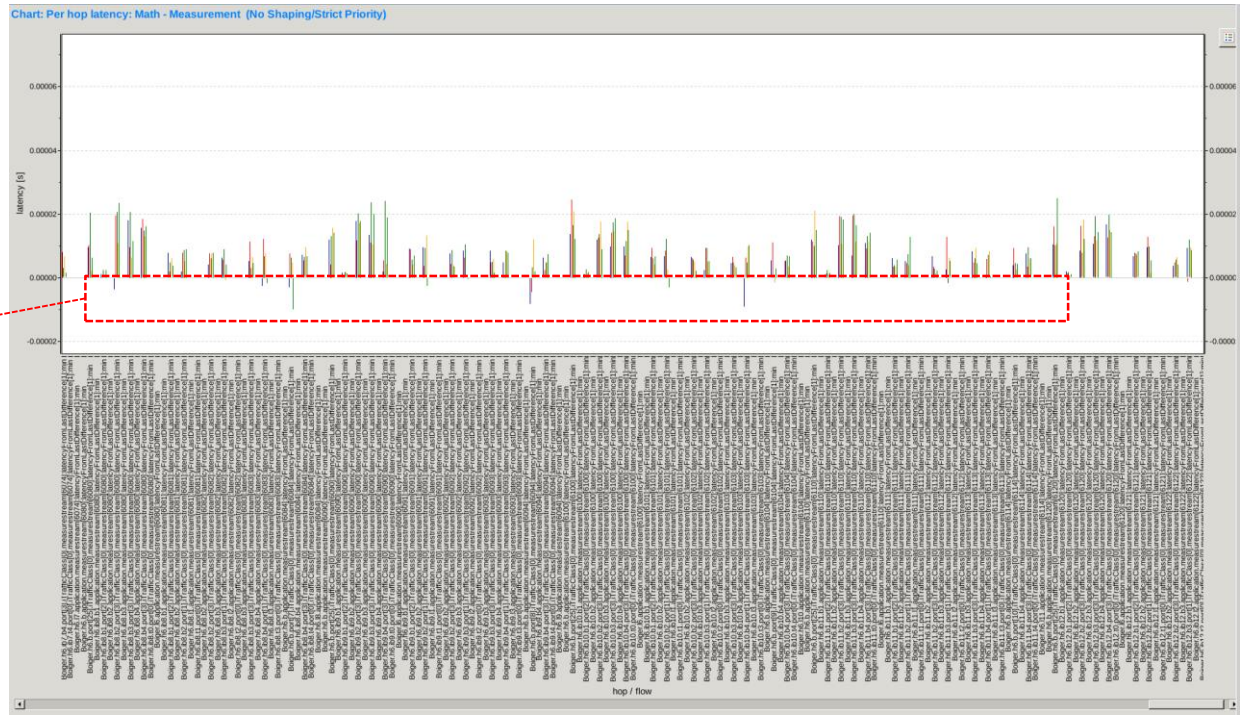


Description

- Per hop/per flow (math-measurement) , minimum over simulation run time
- Simulation shows UBS performance with shared queues

Results: Per hop Latency, Math vs. Measurement (proposed UBS Algorithm)

These are the bad guys! One is enough to show something is wrong…

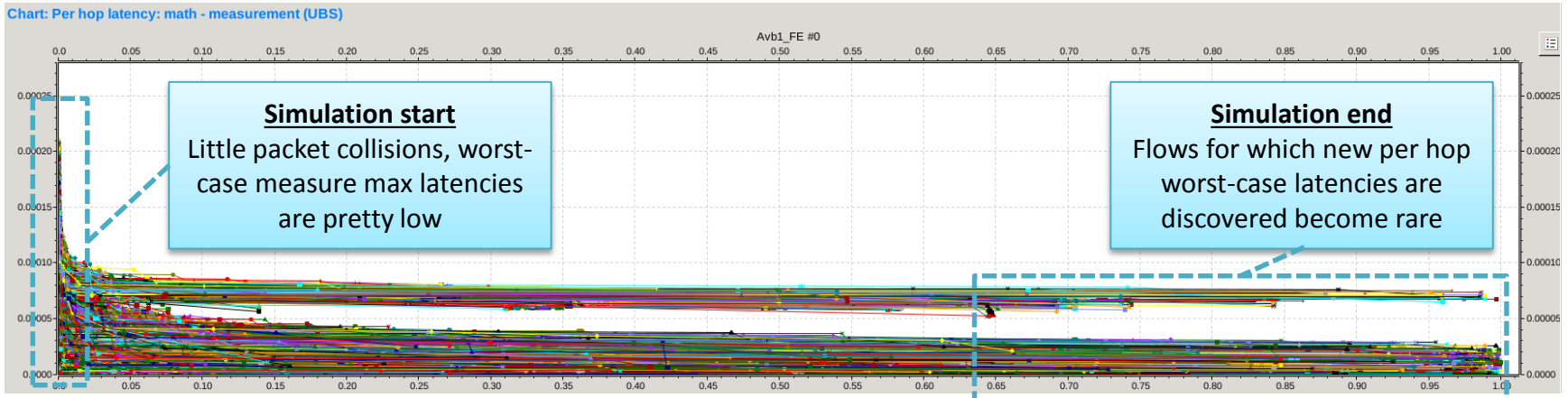


Description

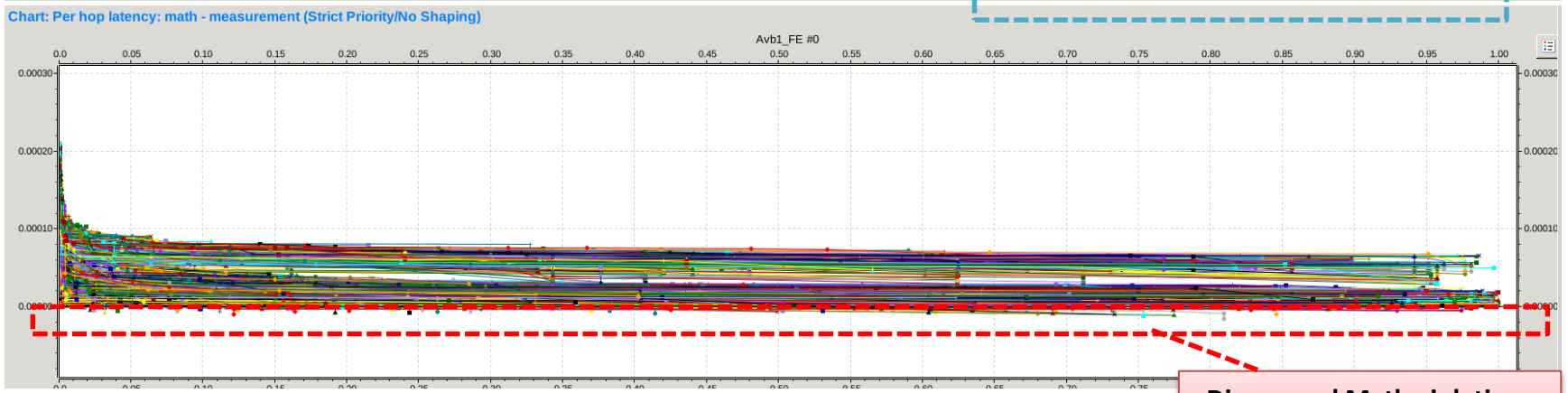
- Per hop/per flow (math-measurement) , minimum over simulation run time
- Simulation shows per hop latency math violations if no shaping would be applied, i.e. this diagram is just for comparison / to check that we‘re doing right

Results: Per Hop Latency Discovered „Low-Score“ over time

Proposed UBS Algorithm



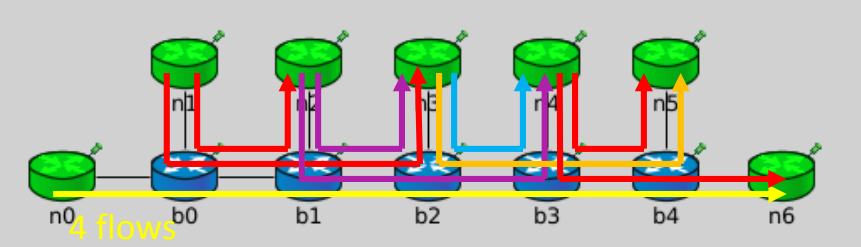
Strict Priority/ No Shaping



Description

- Diagrams capture the new min(math-measurement) per flow over time for all runs
- Still busy on the right, i.e. many new max. per hop latency values are still discovered (simulation limits)
- Likely that better simulation machines would discover more bad guys

Synthetic Benchmarks

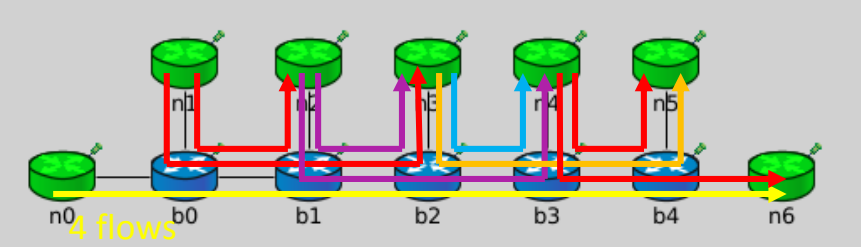


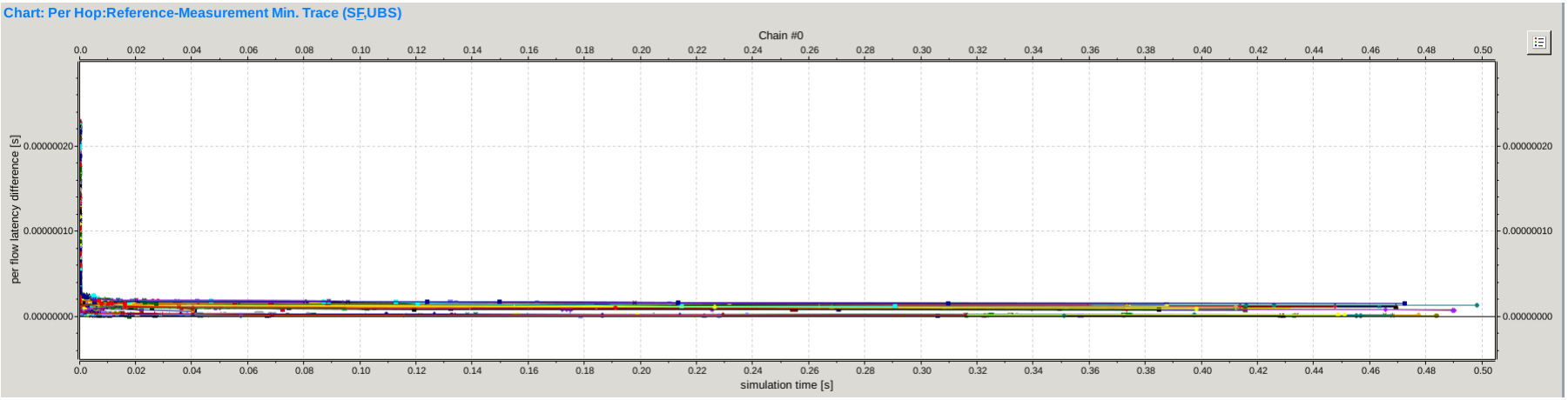
Description

- 4 “poor” flows along the same path need to get through interferred ports
- Two sent by each top node (n1,…,n4) to the next two nodes
- 99% link speed reserved for UBS flows at peak ports
- Different runs of the same experiment (all combinations):
  - Cut-Through & Store-and-Forward
  - Talkers send flows in busy-phases at 100% reserved bandwidth, variable packet size
  - Gaps of variable length between the busy-phases (10% of busy duration in average)
  - Multiple phase duration setups

```
**.burstDurationExpr    = ${bd="uniform(1us,5us)",    "uniform(10us,50us)","uniform(100us,500us)","uniform(1000us,5000us)","uniform(10000us,50000us)"}
**.gapDurationExpr      = ${gd="uniform(0.1us,0.5us)","uniform(1us,5us)",  "uniform(10us,50us)",  "uniform(100us,500us)"  ,"uniform(1000us,5000us)" ! bd}
```

Results: Per Hop Worst-Case Latency over Time (proposed UBS Algorithm)

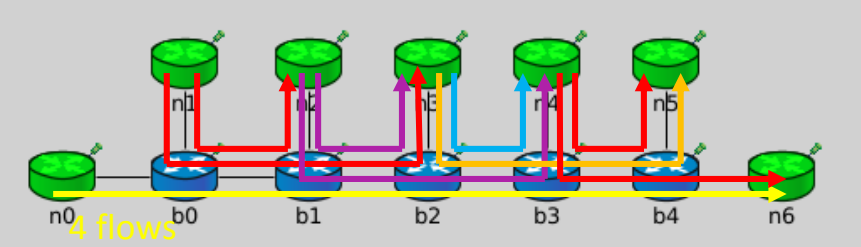


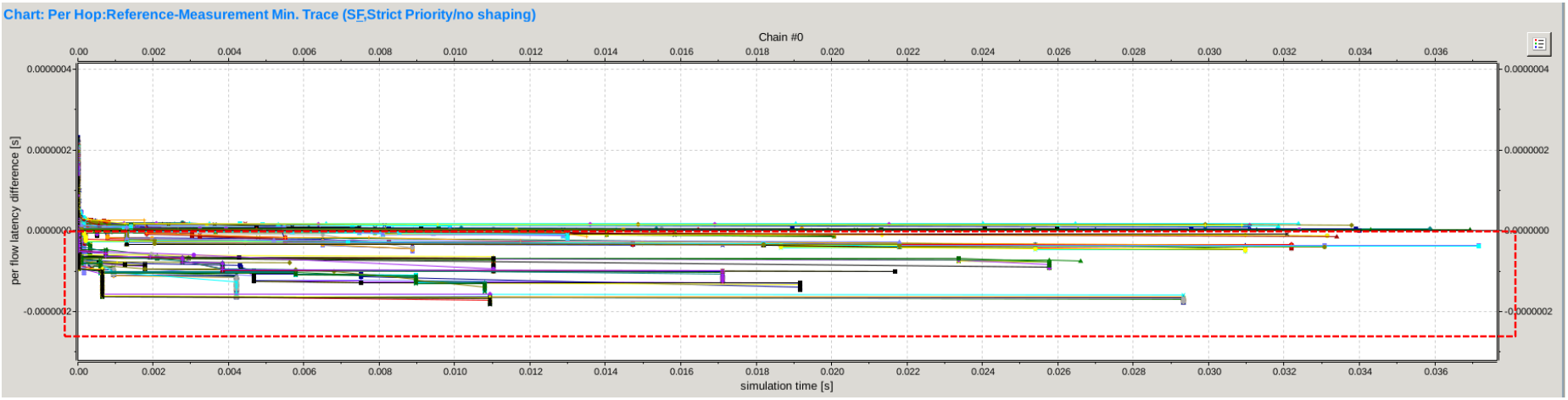


Description

- No violations of the per hop latency math, i.e. no negative values
- Density of discoverd values decreases visually for the given simulation runtime of 0.5 seconds on the right

Results: Per Hop Worst-Case Latency over Time (No Shaping)

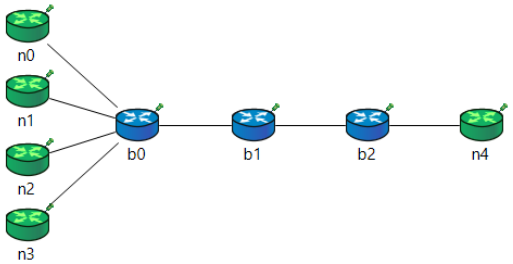


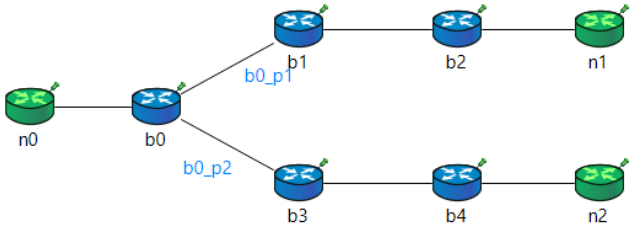


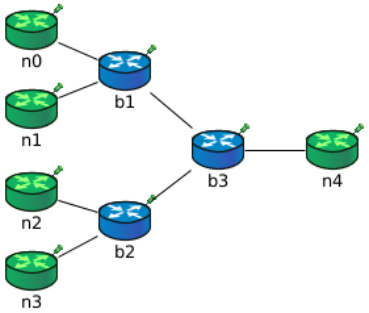
Description

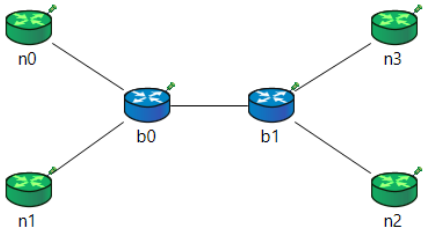
- The same simulation, but without shaping (for comparison)
- This synthetic benchmark is more aggressive to discover per hop latency violations

More Synthetic Benchmarks: Atomic Scenarios









Description

- Multiple flows running from left to right (max. packet length ≤ 10 bit)
- Best Effort interference: 1..200 bit Packets (varying at runtime)
- Results are basically the same (no violations)

Summary

This slide set proposed a UBS derivate

- Maintains essential UBS features
  - High Link Utilization
  - Low Latency Guarantees
  - Protection
  - No clock sync. Dependencies
- Provides device specific queue limits:
  - (Port Count – 1) is enough for small devices (mandatory),
  - More enables sub-priorities (implementer decision)

  →Solves the scalability concerns
- Latency math verified by various simulations

Ongoing

- Harden proof of the math
- Math taking speedup into account
- Other?!

Thank you for your Attention!

Questions, Opinions, Ideas?

Johannes Specht

Dipl.-Inform. (FH)

Dependability of Computing Systems
Institute for Computer Science and
Business Information Systems (ICB)
Faculty of Economics and
Business Administration
University of Duisburg-Essen

Schuetzenbahn 70
Room SH 502
45127 Essen
GERMANY
T +49 (0)201 183-3914
F +49 (0)201 183-4573

Johannes.Specht@uni-due.de
http://dc.uni-due.de