## Automatic In-network Control Empowered by Programmable Infrastructure

Liangcheng Yu

#### 10/07/2022 @MSR Networking Research Group



#### Ubiquitous network control tasks



### Ubiquitous network control tasks

#### **Out-of-control events**

- Congestion collapse
- TCP incast
- Network hotspot
- DoS attack
- Network failure
- Time drift

. . .

Bandwidth starvation



## Ubiquitous network control tasks

#### **Out-of-control events**

- Congestion collapse
- TCP incast
- Network hotspot
- DoS attack
- Network failure
- Time drift

. . .

Bandwidth starvation



#### Closed-loop control mechanisms

- Congestion control
- Desynchronization
- Load balancing
- Defense policy
- Failure mitigation
- Clock synchronization
- Fairness control

. . .





#### **System identification**



#### **Measurement**

Target signals? Granularity? Explicit or implicit? Synchronous?

#### System identification



#### **Measurement**

Target signals? Granularity? Explicit or implicit? Synchronous?

#### **Controller logic**

Position? Distributed? Stability? Control interval time scale?

#### System identification

#### Actuation

Adaptive? Rate limiter config? Switch weights? Routes? Pacing?

#### **Controller logic**

Position? Distributed? Stability? Control interval time scale?



#### **Measurement**

Target signals? Granularity? Explicit or implicit? Synchronous?

#### System identification



Faster networks  $< 1 \rightarrow 10 \rightarrow 100 \rightarrow 800 \rightarrow ...[Gbps]$ 



Faster networks  $< 1 \rightarrow 10 \rightarrow 100 \rightarrow 800 \rightarrow ...[Gbps]$ 

- *Microscopic* (*O*(*µs*)) events are prevalent
- Challenging to sense, analyze, and react



Faster networks  $< 1 \rightarrow 10 \rightarrow 100 \rightarrow 800 \rightarrow ...[Gbps]$ 

- *Microscopic* (*O*(*µs*)) events are prevalent
- Challenging to sense, analyze, and react

Traditional network control

• Infrequent (O(100 ms)), asynchronous, and manual



Faster networks  $< 1 \rightarrow 10 \rightarrow 100 \rightarrow 800 \rightarrow ...[Gbps]$ 

- *Microscopic* (*O*(*µs*)) events are prevalent
- Challenging to sense, analyze, and react

Traditional network control

• Infrequent (O(100 ms)), asynchronous, and manual

Towards fast, real-time, and automatic control at scale?





- High throughput (>12.8 Tbps, billions of operations/s)
- Little additional overhead with processing logic



#### **Specialization**

- High throughput (>12.8 Tbps, billions of operations/s)
- Little additional overhead with processing logic

#### Locational benefits

- Customizable line-rate processing to reduce (tail) latency
- Visibility into accurate network information

Once you've got a software platform where you can **change its behavior**, you can start introducing previously absurd-sounding ideas, including fanciful ideas of **automatic, real-time, closedloop control of an entire network**." — Nick McKeown

- 12.8 Tbps of throughput and beyond
- Low overhead with additional processing logic
- **Unique locational benefits** 
  - Processing along packet path to reduce (tail) latency
  - Visibility into accurate network information











# Outline



#### OrbWeaver: Using IDLE Cycles in Programmable Networks for Opportunistic Coordination



• A primary goal of computer networks: *deliver packets* 

- A primary goal of computer networks: *deliver packets* 
  - User application: video streaming, web browsing, file transfer...

- A primary goal of computer networks: *deliver packets* 
  - **User application:** video streaming, web browsing, file transfer...
  - Non-user application: control messages, probes about network state, keep alive heartbeats...

- A primary goal of computer networks: *deliver packets* 
  - User application: video streaming, web browsing, file transfer...
  - Non-user application: control messages, probes about network state, keep alive heartbeats...
- Often, two classes of traffic *multiplex* the same network

To consume extra BW for fidelity (of the control application), or not to?

To consume *extra BW* for *fidelity* (of the control application), or not to?

• Time synchronization: **clock-sync rate**  $\rightarrow$  **clock precision** 

To consume extra BW for fidelity (of the control application), or not to?

- Time synchronization: **clock-sync rate**  $\rightarrow$  **clock precision**
- Failure detector: keep alive message frequency → detection speed
- Congestion notification: **signaling data/rate** → **measurement accuracy**
- In-band telemetry: INT postcard volume → post-mortem analysis



To consume extra BW for fidelity (of the control application), or not to?

- *Time synchronization*: **clock-sync rate** → **clock precision**
- Failure detector: keep alive message frequency → detection speed
- Congestion notification: **signaling data/rate** → **measurement accuracy**
- *In-band telemetry*: **INT postcard volume** → **post-mortem analysis**

Is the trade-off between fidelity and overhead necessary?

To consume extra BW for fidelity (of the control application), or not to?

- - Can we coordinate at **high-fidelity** with a **near-zero cost** (to usable bandwidth, latency...)?
- In-band telemetry: IN I postcard volume  $\rightarrow$  post-mortem analysis

Is the trade-off between fidelity and overhead necessary?

#### Can we coordinate at **high-fidelity** with a **near-zero cost** to usable bandwidth and latency?

#### Idea: Weaved Stream

- Exploit *every gap* (O(100ns)) between user packets opportunistically
- Inject customizable IDLE packets carrying information across devices

## Opportunity: $< \mu s$ gaps are prevalent



•

. . .

Structural asymmetry



• Union of **user** and **IDLE** (injected) packets:





• Union of **user** and **IDLE** (injected) packets:



**[R1 Predictability]** Interval between **any two consecutive** packets  $\leq \tau$


• Union of **user** and **IDLE** (injected) packets:



[R1 Predictability] Interval between any two consecutive packets  $\leq \tau$ [R2 Little-to-zero overhead] Not impact user packets or power draw



• Union of upper and IDLE (inicated) packates

#### Implement many *in-network applications* (failure detection, clock sync, congestion notification...) *for free!*

**[R1 Predictability]** Interval between **any two consecutive** packets  $\leq \tau$ **[R2 Little-to-zero overhead]** Not impact user packets or power draw



• Union of **user** and **IDLE** (injected) packets:



Extending IDLE characters to higher layers

Crazy idea?

- Data plane packet generator
- Replication engine
- Data plane programmability
- Flexible switch configuration (priorities, buffers...)

[R1 Predicta [R2 Little-to

## Outline

- 1. Switch data plane architecture
- 2. Weaved stream generation
- 3. OrbWeaver applications

### RMT switch model



## Naive weaved stream generation



## Naive weaved stream generation



Predictability even there is no user traffic 🕢

## Problems with blind injection



**Scalability:** overwhelm packet generator capacity to satisfy target rate

## Problems with blind injection



**Scalability**: overwhelm packet generator capacity to satisfy target rate

**Interference upon cross-traffic:** throughput, latency, or loss of user traffic!

### Amplify seed stream



## Amplify seed stream



Monopolize usage and waste PRE packet-level BW!

## Amplify seed stream on demand



#### Selective filtering

- (Tiny) sending history state of past cycle to each egress port
- Create an IDLE packet to a port *only if we need an IDLE packet*

## Amplify seed stream on demand



#### Selective filtering

- (Tiny) sending history state of past cycle to each egress port
- Create an IDLE packet to a port *only if we need an IDLE packet*

#### Cross-traffic contention



#### Cross-traffic contention



### Cross-traffic contention



## Preventing contention



**Rich configuration options for priorities and buffer management** 

## Preventing contention



#### **Rich configuration options for priorities and buffer management**

- Zero impact of weaved stream predictability
- Zero impact of user traffic throughput or buffer usage

## Preventing contention



#### **Rich configuration options for priorities and buffer management**

- Zero impact of weaved stream predictability
- Zero impact of user traffic throughput or buffer usage
- Negligible impact of latency of user packets

## Implementation and evaluation

Hardware prototype on a pair of Wedge100BF-32X Tofino switches



# *Takeaway*: Little-to-no impact of power draw, latency, or throughput while guaranteeing **predictability** of the weaved stream!

Hardware prototype on a pair of Wedge100BF-32X Tofino switches



#### OrbWeaver use cases









## OrbWeaver use cases





Performance aware routing Flowlet load imbalance Consistent replicas Latency Network queries localization Header compression Microburst detection In-band telemetry Self-healing failure detection Network queries Event-based network control Clock synchronization Packet forensics

## OrbWeaver use cases



*Fine-grained network state inference [R1]* 

Clock synchronization

Performance aware routingFlowlet load imbalanceConsistent replicasNetwork queriesLatency<br/>localizationHeader compressionMicroburst detectionIn-band telemetryEvent-based<br/>network controlSelf-healing failure detectionNetwork queries

Packet forensics



Traditional two-way protocol



Traditional two-way protocol

Existing approaches for high precision

- Require special hardware (such as DTP)
- Require messaging overheads (such as DPTP)



Traditional two-way protocol

Existing approaches for high precision

- Require special hardware (such as DTP)
- Require messaging overheads (such as DPTP)

Challenges to achieve ns precision

- Messaging frequency v.s. clock precision
- Inaccuracies due to queueing delays





1. Embed timestamp information in **free IDLE packets** [R2]



- 1. Embed timestamp information in **free IDLE packets** [R2]
- 2. Selective synchronization: **infer queue delay** from IDLE gaps and filter out **unreliable messages** [R1]



- 1. Embed timestamp information in **free IDLE packets** [R2]
- 2. Selective synchronization: **infer queue delay** from IDLE gaps and filter out **unreliable messages** [R1]



Achieve same or better performance with close-to-zero overheads

## Example: failure detection



Common approaches

- Regular, high priority heartbeats
- Conservative detection thresholds

Traditional uni-directional heartbeats

## Example: failure detection



Traditional uni-directional heartbeats

#### Common approaches

- Regular, high priority heartbeats
- Conservative detection thresholds



- Exploit **guaranteed predictability** of max gap with weaved stream (e.g., 120 ns for 100GbE) [R1]
- Exploit **IDLE bandwidth** to send flow event logs [R2]

## Example: failure detection



Traditional uni-directional heartbeats

#### Common approaches

- Regular, high priority heartbeats
- Conservative detection thresholds



- Exploit **guaranteed predictability** of max gap with weaved stream (e.g., 120 ns for 100GbE) [R1]
- Exploit **IDLE bandwidth** to send flow event logs [R2]

Push toward an instantaneous **self-healing failure mitigation** (with data plane reroute) of **optional speed** 



- Weaved stream abstraction to harvest IDLE cycles
  - Guarantee predictability with little-to-zero overhead


- Weaved stream abstraction to harvest IDLE cycles
  - Guarantee predictability with little-to-zero overhead
- Generic support of a wide range of data plane applications for free
  - **Don't** need to choose between coordination fidelity and bandwidth overhead

## https://github.com/eniac/OrbWeaver

# Outline



## OrbWeaver: Using IDLE Cycles in Programmable Networks for Opportunistic Coordination



# Public networks care about fairness



# Public networks care about fairness



# Public networks care about fairness



# Fairness enforcement at the end hosts?



# Fairness enforcement at the end hosts?



# Fairness enforcement at the end hosts?





- Existing approaches suffer from limited practicalities
  - **Assumption**: specialized hardware for per-flow queues, end-host cooperation...

- Existing approaches suffer from limited practicalities
  - **Assumption**: specialized hardware for per-flow queues, end-host cooperation...
- AFQ [NSDI '18]: practical emulation of ideal FQ on COTS hardware
  - **Constraints**: e.g., # priorities, # queues, buffers

- Existing approaches suffer from limited practicalities
  - **Assumption**: specialized hardware for per-flow queues, end-host cooperation...
- AFQ [NSDI '18]: practical emulation of ideal FQ on COTS hardware
  - **Constraints**: e.g., # priorities, # queues, buffers

Challenging to strictly enforce FQ on each individual flow



- Relaxation of fairness at every instance in time
  - Penalize/redistribute BW from flows exceeding fair share to others



- Relaxation of fairness at every instance in time
  - Penalize/redistribute BW from flows exceeding fair share to others
- Binary classification of flows
  - Efficiently implement various subroutines (e.g., leaky-bucket filter)



- Relaxation of fairness at every instance in time
  - Penalize/redistribute BW from flows exceeding fair share to others
- Binary classification of flows
  - Efficiently implement various subroutines (e.g., leaky-bucket filter)

### **Cebinae router architecture for binary taxation**

- Zero modifications and coordinations to/with legacy host CCAs
- Requirement of only two queues/priorities
- Compatibility with CCAs operating on both loss and delay signals

# Outline

### 1. Conceptual foundation for binary classification

2. Cebinae's taxation mechanism

3. Evaluation

# Max-min fairness



# Max-min fairness



Implication: distributed verification of max-min fairness

Each link *l* can determine the set of *bottlenecked* flows:

Each link *l* can determine the set of *bottlenecked* flows:

### If *l* non-saturated:

All flows not bottlenecked

Each link *l* can determine the set of **bottlenecked** flows:

## If *l* non-saturated:

All flows not bottlenecked

Else, for each flow *i*:

If *i* is among *l*'s largest rate(s)

*i* is **bottlenecked** at *l* 

Else

*i* is **not bottlenecked** at *l* 

Each link *l* can determine the set of **bottlenecked** flows:

## If *l* non-saturated:

All flows not bottlenecked

Else, for each flow *i*:

If *i* is among *l*'s largest rate(s)

*i* is **bottlenecked** at *l* 

### Else

*i* is **not bottlenecked** at *l* 

### **Observation:**

1. Each *conditional* can be determined using **only local information** 

Each link *l* can determine the set of **bottlenecked** flows:

## If *l* non-saturated:

All flows not bottlenecked

Else, for each flow *i*:

If *i* is among *l*'s largest rate(s)

*i* is **bottlenecked** at *l* 

### Else

*i* is **not bottlenecked** at *l* 

#### **Observation:**

- 1. Each *conditional* can be determined using **only local information**
- 2. Binary classification: bottlenecked  $(\top)$ , not bottlenecked  $(\bot)$

# Naive enforcement

Each link *l* can determine the set of bottlenecked flows:

If *l* non-saturated:

NOP

### Else, for each flow *i*:

### If *i* is among *l*'s largest rate(s)

Drop packets of **all** is per their current rate

### Else

#### NOP

# Naive enforcement

Each link *l* can determine the set of bottlenecked flows:

If *l* non-saturated:

NOP

### Else, for each flow *i*:

### If *i* is among *l*'s largest rate(s)

Drop packets of **all** is per their **current rate** 

#### Else

NOP

### **Drawbacks:**

1. Can't make an **alreadyunfair** allocation fair

# Naive enforcement

Each link *l* can determine the set of bottlenecked flows:

If *l* non-saturated:

NOP

### Else, for each flow *i*:

If *i* is among *l*'s largest rate(s)

Drop packets of all is per their current rate

Else

NOP

### **Drawbacks:**

1. Can't make an **alreadyunfair** allocation fair

# 2. CCAs may not be responsive to **loss signals**



Each link *l* can determine the set of bottlenecked flows:

If *l* non-saturated:

NOP

### Else, for each flow *i*:

If *i* is among *l*'s largest rate(s)

**Penalize all is per their current rate** 

#### Else

NOP

1. **Penalty box** with non-loss signals such as delay, ECN



Each link *l* can determine the set of bottlenecked flows:

If *l* non-saturated:

NOP

### Else, for each flow *i*:

If *i* is among *l*'s largest rate(s)

Penalize all is per their taxed rate

#### Else

NOP

- 1. **Penalty box** with non-loss signals such as delay, ECN
- 2. Collective redistribution of bandwidth to  $\perp$  flows

# Tax bottlenecked-flows exceeding

#### fair bandwidth share Each link t can determine the set of bottlenecked flows:



Penalty box with non-loss signals such as delay
Collective redistribution of bandwidth to ⊥ flows

# Tax bottlenecked-flows exceeding



# Instantiation: Cebinae router architecture



# Instantiation: Cebinae router architecture



# Instantiation: Cebinae router architecture



## Normal operation



## Normal operation



## Normal operation


#### Normal operation



#### Normal operation



#### Normal operation









Virtual pacing : guarantee no reordering and avoid violation of draining deadline in the worst case



**Atomic transactions**: LBF states and egress caches

# Implementation and evaluation

Hardware prototype on a Wedge100BF Tofino switch testbed and NS-3 module

- Is Cebinae agnostic to CCAs?
- Can Cebinae mitigates unfairness (RTT, inter-CCA)?
- Can Cebinae move towards max-min fairness?
- Is Cebinae easy to configure?
- Does Cebinae resource usage scale?









Mitigates the **skewed and persistent unfairness** with little efficiency impact: **JFI from 0.093 to 0.984** 





Preventing aggressiveness

Mitigating starvation



				Throughput [Mbps]			Goodput [Mbps]			JFI		
Btl. BW	RTTs [ms]	Buf. [MTU]	CCAs	FIFO	FQ	Cebinae	FIFO	FQ	Cebinae	FIFO	FQ	Cebinae
100 Mbps	{20.8, 28}	250	{NewReno:2, NewReno:8}	98.95	95.62	95.92	95.35	92.16	92.44	0.740	0.982	0.999
100 Mbps	{20.4, 40}	350	{Cubic:8, Cubic:2}	98.96	98.95	98.00	95.37	95.37	94.45	0.539	1.000	0.980
100 Mbps	{20.4, 60}	500	{Vegas:2, Vegas:8}	98.88	98.83	98.88	95.29	95.24	95.29	0.873	1.000	0.993
100 Mbps	{200}	1700	{NewReno:16, Cubic:1}	98.28	90.99	94.53	94.38	87.61	91.02	0.446	0.995	0.925
100 Mbps	{100}	850	{NewReno:16, Cubic:1}	98.72	91.45	95.58	95.11	88.10	92.08	0.857	0.998	0.960
100 Mbps	<b>{50}</b>	420	{NewReno:16, Cubic:1}	98.90	93.86	95.37	95.30	90.45	91.90	0.936	0.999	0.993
100 Mbps	<b>{50}</b>	420	{Vegas:16, Cubic:1}	98.90	98.90	95.47	95.30	95.30	91.99	0.096	1.000	0.988
100 Mbps	{100}	850	{Vegas:16, NewReno:1}	98.71	97.77	95.67	95.07	94.19	92.16	0.093	0.999	0.985
100 Mbps	{100}	850	{Vegas:128, NewReno:1}	98.88	98.74	97.45	95.26	95.10	93.88	0.189	0.966	0.976
100 Mbps	<b>{60}</b>	500	{Vegas:8, NewReno:8, Cubic: 2}	98.87	98.02	96.52	95.27	94.45	93.00	0.510	0.991	0.973
1 Gbps	{5}	420	{NewReno:32, Cubic:8}	989.8	989.8	985.4	954.0	954.0	949.7	0.844	0.988	0.955
1 Gbps	{10}	850	{Vegas:128, Cubic:1}	989.8	989.8	968.0	954.0	954.0	932.9	0.048	0.966	0.953
1 Gbps	{10}	850	{Vegas:1024, Cubic:2}	989.8	989.8	949.2	953.6	953.6	914.1	0.275	0.833	0.846
1 Gbps	<b>{50}</b>	4200	{NewReno: 128, BBR: 1}	988.7	923.6	981.6	952.7	890.0	945.8	0.992	0.975	0.990
1 Gbps	<b>{50}</b>	4200	{NewReno: 128, BBR: 2}	988.9	953.9	979.9	952.8	919.2	944.2	0.951	0.963	0.981
1 Gbps	<b>{50}</b>	21000	{NewReno: 128, BBR: 2}	988.8	953.9	963.8	952.7	919.2	928.7	0.773	0.963	0.936
1 Gbps	{100}	8350	{NewReno: 128, BBR: 2}	986.9	938.2	956.3	950.7	903.9	921.1	0.884	0.968	0.967
1 Gbps	{10}	850	{Vegas:64, NewReno:1}	989.8	989.8	976.2	953.8	954.0	940.7	0.042	0.967	0.976
1 Gbps	{100}	8500	{Vegas:4, NewReno:128}	986.9	917.6	957.3	950.8	884.1	922.2	0.946	0.970	0.971
1 Gbps	{100, 64}	8500	{Vegas:4, NewReno:128}	988.4	941.1	959.8	952.4	906.8	924.7	0.956	0.970	0.964
1 Gbps	{100}	8500	{Vegas:8, NewReno:128}	987.0	936.1	964.4	950.8	901.8	929.0	0.921	0.968	0.969
1 Gbps	{10}	850	{Vegas:128, BBR:1}	989.8	989.8	987.3	954.0	954.0	951.5	0.886	0.965	0.985
1 Gbps	<b>{100}</b>	8500	{Bic:2, Cubic:32}	985.1	960.3	952.6	944.9	924.9	911.3	0.799	0.999	0.946
10 Gbps	<b>{50, 44}</b>	41667	{NewReno:128, Cubic:16}	9876	9705	9780	9514	9352	9420	0.917	0.969	0.968
10 Gbps	{28, 28}	25000	{NewReno:128, Cubic:128}	9891	9856	9787	9532	9498	9432	0.863	0.942	0.952

# Cebinae is agnostic to CCAs

			13 Marine Marine Harrison and the state of the second second second second second second second second second s	Throughput [Mbps]			Goodput [Mbps]			JFI		
Btl. BW	RTTs [ms]	Buf. [MTU]	CCAs	FIFO	FQ	Cebinae	FIFO	FQ	Cebinae	FIFO	FQ	Cebinae
100 Mbps	{20.8, 28}	250	{NewReno:2, NewReno:8}	98.95	95.62	95.92	95.35	92.16	92.44	0.740	0.982	0.999
100 Mbps	$\{20.4, 40\}$	350	{Cubic:8, Cubic:2}	98.96	98.95	98.00	95.37	95.37	94.45	0.539	1.000	0.980
100 Mbps	{20.4, 60}	500	{Vegas:2, Vegas:8}	98.88	98.83	98.88	95.29	95.24	95.29	0.873	1.000	0.993
100 Mbps	{200}	1700	{NewReno:16, Cubic:1}	98.28	90.99	94.53	94.38	87.61	91.02	0.446	0.995	0.925
100 Mbps	<b>{100}</b>	850	{NewReno:16, Cubic:1}	98.72	91.45	95.58	95.11	88.10	92.08	0.857	0.998	0.960
100 Mbps	<b>{50}</b>	420	{NewReno:16, Cubic:1}	98.90	93.86	95.37	95.30	90.45	91.90	0.936	0.999	0.993
100 Mbps	<b>{50}</b>	420	{Vegas:16, Cubic:1}	98.90	98.90	95.47	95.30	95.30	91.99	0.096	1.000	0.988
100 Mbps	<b>{100}</b>	850	{Vegas:16, NewReno:1}	98.71	97.77	95.67	95.07	94.19	92.16	0.093	0.999	0.985
100 Mbps	<b>{100}</b>	850	{Vegas:128, NewReno:1}	98.88	98.74	97.45	95.26	95.10	93.88	0.189	0.966	0.976
100 Mbps	<b>{60}</b>	500	{Vegas:8, NewReno:8, Cubic: 2}	98.87	98.02	96.52	95.27	94.45	93.00	0.510	0.991	0.973
1 Gbps	{5}	420	{NewReno:32, Cubic:8}	989.8	989.8	985.4	954.0	954.0	949.7	0.844	0.988	0.955
1 Gbps	{10}	850	{Vegas:128, Cubic:1}	989.8	989.8	968.0	954.0	954.0	932.9	0.048	0.966	0.953
1 Gbps	{10}	850	{Vegas:1024, Cubic:2}	989.8	989.8	949.2	953.6	953.6	914.1	0.275	0.833	0.846
1 Gbps	<b>{50}</b>	4200	{NewReno: 128, BBR: 1}	988.7	923.6	981.6	952.7	890.0	945.8	0.992	0.975	0.990
1 Gbps	<b>{50}</b>	4200	{NewReno: 128, BBR: 2}	988.9	953.9	979.9	952.8	919.2	944.2	0.951	0.963	0.981
1 Gbps	<b>{50}</b>	21000	{NewReno: 128, BBR: 2}	988.8	953.9	963.8	952.7	919.2	928.7	0.773	0.963	0.936
1 Gbps	{100}	8350	{NewReno: 128, BBR: 2}	986.9	938.2	956.3	950.7	903.9	921.1	0.884	0.968	0.967
1 Gbps	{10}	850	{Vegas:64, NewReno:1}	989.8	989.8	976.2	953.8	954.0	940.7	0.042	0.967	0.976
1 Gbps	<b>{100}</b>	8500	{Vegas:4, NewReno:128}	986.9	917.6	957.3	950.8	884.1	922.2	0.946	0.970	0.971
1 Gbps	{100, 64}	8500	{Vegas:4, NewReno:128}	988.4	941.1	959.8	952.4	906.8	924.7	0.956	0.970	0.964
1 Gbps	<b>{100}</b>	8500	{Vegas:8, NewReno:128}	987.0	936.1	964.4	950.8	901.8	929.0	0.921	0.968	0.969
1 Gbps	{10}	850	{Vegas:128, BBR:1}	989.8	989.8	987.3	954.0	954.0	951.5	0.886	0.965	0.985
1 Gbps	<b>{100}</b>	8500	{Bic:2, Cubic:32}	985.1	960.3	952.6	944.9	924.9	911.3	0.799	0.999	0.946
10 Gbps	{50, 44}	41667	{NewReno:128, Cubic:16}	9876	9705	9780	9514	9352	9420	0.917	0.969	0.968
10 Gbps	{28, 28}	25000	{NewReno:128, Cubic:128}	9891	9856	9787	9532	9498	9432	0.863	0.942	0.952

# Summary



- No modifications nor coordinations to/with legacy host CCAs
  - Real-time switch architecture serializing in-network compute modules
- COTS hardware and **minimal resource requirements** 
  - Two queues/priorities are sufficient
- Compatible with CCAs using both loss and non-loss signals
  - Generic support of a wide range of Internet CCAs and environments



#### More details



# Q & A

# Using weaved stream



# Optimal value of $\tau$

$$\tau = B_{100Gbps} / MTU_{1500B} = 120ns$$
100 Gbps
$$t$$