Gao, P.X. *et al.*,
"pHost: Distributed near-optimal datacenter transport over commodity network fabric"
Proc. of ACM CoNEXT '15, 2015.

Ming zhi Yu
Zaina Hamid

This paper talks about:

pHost : a transport design aimed at minimizing FCT, by achieving

- Near-optimal performance of pFabric & commodity network design of Fastpass
- While overcoming Fastpass' overheads

Evaluates pHost and its **requirements**, **design** attributes, and gives an insight into its **test setup**, **performance** & **metric evaluation**, and overall applicability.

Introduction

- Optimize network performance in data centers, hence focus on FCT
- Currently pFabric achieves close to theoretically minimal slowdown over workloads, However for more optimization
  - pFabric needs specialised network hardware to implement a specific packet scheduling and queue management algorithm
- Fastpass uses commodity switches coupled with a central scheduler, BUT performance is significantly worse

SOLUTION: pHost

- No specialized n/w h/w, no per flow state or complex rate calculations
- No centralized scheduler, no explicit n/w feedback

pHost overview

Modern DataCenter Networks:

- Small RTTs - due to lower latencies
- Full bisection bandwidth - Due to advanced datacenter network topologies
- Simple switches - Basic features such as priority levels, ECMP and/or packet spraying, cut through switching, and relatively small buffers

Basic Transport Mechanism

1. Source sends **RTS** packet to destination
2. Per packet transmission time, the destination considers the set of pending RTSs and sends a **"token"** to a corresponding host.
   a. Hence the source transmits one data packet from that flow at a specific **priority level**
   b. per-packet scheduling across active flows independent of other destinations
3. Tokens can expire if not used in a certain time by the source
   a. Source has a few **"free tokens"** for each flow by default
4. Source **chooses** an unexpired token and sends corresponding data packet, hence selecting across active flows independent of other sources
5. Destination sends ACK on receiving all packets for a flow

*Control packets have the highest priority
**Degree of freedom can be configured to achieve different optimizations

Why pHost works

- Using packet spraying technique
  - Eliminating core congestion & sophisticated path level scheduling (Fastpass) / detailed packet scheduling in core switches (pFabric)
- Destination congestion:
  - Granting tokens in response to RTS requests
  - Instead of a centralized scheduler, we use a fully decentralized scheduler
- To avoid starvation at host: allow source to launch multiple RTSs in parallel
  - Small budget of free tokens for each flow
  - Send without waiting for the RTT from destination
- To avoid starvation at destination: back-off mechanism
  - Destination avoids sending tokens to the source if previously sent tokens were unused

Design Details

For token assignment and utilization - SOURCE

1. RTS includes other information: flow size, deadline, tenant the flow belongs to, etc (used by destination)
1. Active Tokens = per flow list of tokens maintained by the source, with each token representing permission to send one packet to the flow's destination. Initially configured w/ set **free tokens**
2. Only if unexpired token is available
3. If idle, sends a packet from the list of available tokens

Algorithm 1 pHost algorithm at Source.

```
if new flow arrives then
    Send RTS
    ActiveTokens ← FreeTokens          ▷ Add free tokens (§3.2)
else if new token T received then
    Set ExpiryTime(T)                  ▷ Tokens expire in fixed time (§3.2)
    ActiveTokens ← T
else if idle then
    T = Pick(ActiveTokens)             ▷ pick unexpired token (§3.3)
    Send Packet corresponding to T
end if
```

For token assignment and utilization - DESTINATION

1. PendingRTSlist = flows for which destination received the RTS but not all the data packets, every new RTS also goes into this list
2. Every MTU the destination selects an RTS from PendingRTS & sends out a token to the source
3. Expired token > threshold, the flow is downgraded for the near future
4. On receiving all packets for a flow, sends an ACK to the source and removes the RTS from the pendingRTS list

*All control packets in pHost are sent at the highest priority

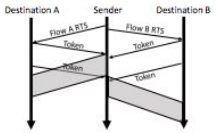
Algorithm 2 pHost algorithm at Destination.

```
if receive RTS R then
    PendingRTS ← R
else if idle then
    F = Pick(PendingRTS)               ▷ Pick an active flow (§3.3)
    Send Token T for flow F
    if F.#ExpiredTokens > Threshold then   ▷ (§3.2)
        Downgrade F for time t*            ▷ (§3.2)
    end if
else if Received data for token T then
    Set Token T as responded
end if
```

Maximizing Network Utilization

Sources sending multiple RTSs in parallel & destinations assigning one token per packet transmission time may lead to network under utilization. Challenges that pHost resolves are:

- Free tokens for new flow arrivals: to overcome waiting until the token for a flow is received, and avoid the effect on short flow performance
- Source downgrading and token expiry: When the source prefers token for Flow B, while the tokens from Flow A are getting wasted, and another source could have interacted with A in the meanwhile - Hence pHost maintains a number of **unexpired tokens** for each source, and when that threshold value is crossed, the source is downgraded for a **timeout** period



Local Scheduling Problem

Goal is to accommodate different performance optimization goals (Eg: optimize tail latency across all flows, or to share n/w b/w fairly among tenants)

- When the source & destination exchange RTS/tokens they also share flow size, deadlines, priority level etc

Performance objectives solved for using end host scheduling:

- **Minimizing FCT**
  - SRPT: flow with least no. of remaining packets is prioritized while assigning tokens
  - Short flows: second highest priority; Long flows: third highest priority
  - Source also prioritizes flows with fewest remaining packets while using tokens & free tokens if any

- **Deadline constrained traffic**
  - EDF: source specifies the deadline in the RTS and the destination prioritizes token assignment accordingly
  - Source also utilize tokens in similar fashion

- **Fairness across multiple tenants**
  - If A runs a web search workload with short flows & B runs a MapReduce workload with long flows
    - pFabric would have prioritised A over B, hence starvation
  - End-host based scheduling: destinations maintain a counter for the no. of packets received from each tenant, & in each unit time assign a token to a flow from the tenant with smaller count
  - Hence fairness across tenants, and also tenant specific performance goals can be met, with specific scheduling policies.

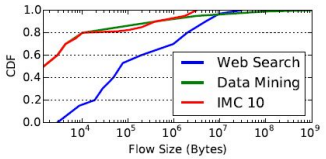
Handling Packet drops

In the rare case of packet drops

- Token is assigned to a specific Packet ID
- If the destination doesn't receive one of the packets until the token has been sent out for the **last** packet of the flow, the destination resends the token for the lost packet ID
- The source retransmits the lost packet(s)

Test setup

- Network topology: two-tier multi-rooted tree with 9 racks and 144 end-hosts.
  - end-host: 10Gbps access link, core switch: nine 40Gbps links
  - Propagation delay: 200ns for each link
  - Queue buffer of each switch port: 6kB - 72kB (default to be 36kB)
- Flows are generated from three production traces
  - “Web Search”, “Data Mining”, and “IMC 10”
  - All three traces are heavy-tailed (most of the flows are short but most of the bytes are in the long flows)
  - Generate flows from traces using a Poisson arrival process with loads ranging from 0.5 to 0.8 (default to be 0.6)



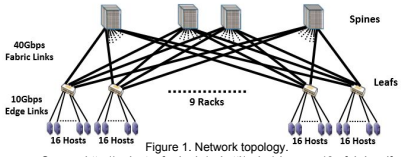


Figure 1. Network topology.
Source: http://web.stanford.edu/~skatti/pubs/sigcomm13-pfabric.pdf

Test setup

- Traffic matrices
  - all-to-all where each source host generates flows to each other host (default)
  - Permutation traffic matrix: each source sends flow to a single destination chosen uniformly at random without replacement
  - Incast traffic matrix: each destination receives flows from a specified number of sources
- Performance metrics
  - Mean *slowdown* := FCT(i) / OPT(i), the smaller the better
    - FCT(i): observed flow completion time when competing with other flows
    - OPT(i): flow completion time of flow i when it is the only flow in the network
  - Normalized FCT := mean of FCT(i) / mean of OPT(i)
  - Throughput := number of bytes delivered to receivers through the network over unit time normalized by the access link bandwidth
  - Fraction of flows that meet their target deadlines
- Evaluated protocols
  - pHost is evaluated against pFabric and Fastpass

Mean slowdown

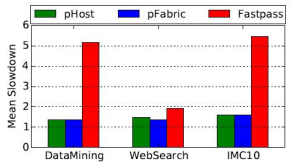
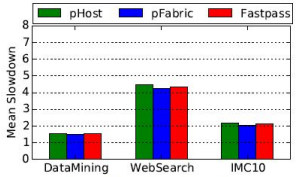


Figure 3: Mean slowdown of pFabric, pHost, and Fastpass across different workloads for our default configuration (0.6 load, per-port buffers of 36kB). pHost performs comparable to pFabric, and 1.3–4× better than Fastpass.

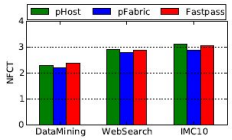


(b) Long flows: Mean Slowdown

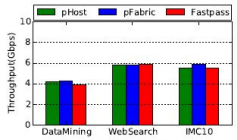
- pFabric achieve near-optimal slowdown, so pHost’s is equally effective at optimizing slowdown
- pHost and pFabric achieve significantly better performance than Fastpass for short flow; all three protocols have comparable performance for long flow.
- Fastpass schedules flows in epochs of 8 packets, so a short flow must wait for at least an epoch, which is around 10us, before it gets scheduled → pHost, pFabric outperform Fastpass in short flow slowdown

Other metrics

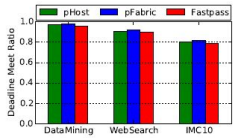
- NFCT
  - all three protocols see similar performance because NFCT is dominated by the FCT of long flows by definition.
- Throughput
  - follow trend similar to NFCT results because overall throughput is dominated by the performance of long flows.
- Deadlines
  - A deadline was assigned to each flow using exponential distribution with mean 100 us (1.25 times its optimal FCT if assigned deadline is less than 1.25 times the optimal FCT)
  - All three protocols offer comparable performance
- Advantages of pHost
  - it relies only on commodity network fabrics
  - Easy to scale because its decentralized controller



(a) NFCT



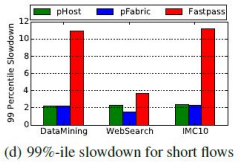
(b) Throughput



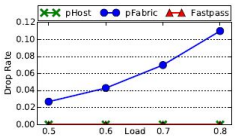
(c) Fraction of flows meeting deadline

Other metrics

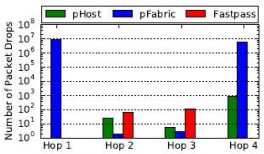
- 99 percentile slowdown
  - 33% higher than the mean slowdown for pHost and pFabric
  - 2 times the mean slowdown for Fastpass
- Drop rate
  - pFabric has higher drop rate because it sends packets more aggressively
  - pHost and Fastpass have close-to-zero drop rate even as load increases
  - Also investigated where packet drops occur by plotting the absolute number of packet drops at each of the 4 hops in the network
    - end-host NIC queue, aggregation switch upstream queue, core switch queue, aggregation switch downstream queue
    - First and last hop drops for pHost and Fastpass are almost eliminated



(d) 99%-ile slowdown for short flows



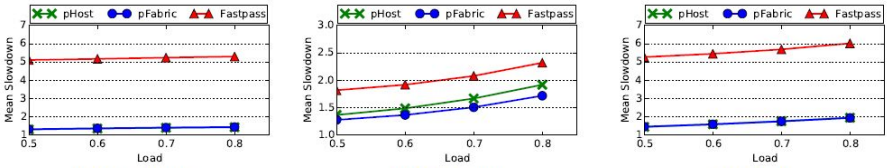
(e) Drop rate



(f) Packet drops across hops

Varying network load

- Evaluated protocol performance for network load varying from 0.5-0.8
- Discovered that relative performance of the different protocols across different network loads remains consistent

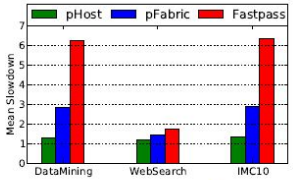


(a) Data Mining (b) Web Search (c) IMC10

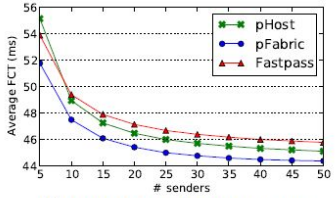
Figure 6: Performance of the three protocols across varying network loads.

Varying traffic matrices

- Permutation TM: overall, pHost performs better than both pFabric and Fastpass
- Incast TM: all three protocols have similar performance for incast ™
  - Total amount of data requested is fixed at 100MB



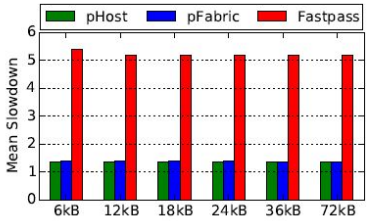
(a) Permutation TM, Figure 2 workloads



(c) Incast TM, flow completion time

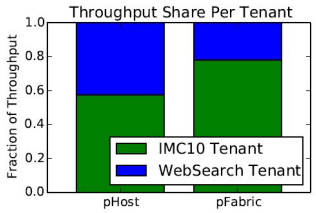
Varying switch parameters

- Evaluated the impact of varying the per-port buffer size in switches
- Evaluated using “Data Mining” workload
- Conclusion: none of the three schemes is sensitive to the sizing of switch buffers



Flexibility -- isolation and fairness between tenants

- pHost can implement arbitrary policies for how tokens are granted and consumed
- Enforced SRPT to prioritize flows with the fewest remaining bytes
- Used two traces “IMC10” and “WebSearch” (mostly short flows)
- pHost provided better isolation and fairness than pFabric



Discussions / Q&A / Comments

- pHost’s advantage wasn’t clearly obvious while comparing against pFabric & Fastpass with respect to additional metrics (NFCT, throughput, & fraction of flows meeting deadlines)
- With respect to websearch, there’s not a significant variation on comparing Fastpass & pHost as expected?
- Packet drops in the case of pHost are higher than pFabric (figure 5f)