

SFC consideration and design

Bottorff, Paul (paul.bottorff@hpe.com)
Chen, Lihao (lihao.chen@huawei.com)
Congdon, Paul (paul.congdon@outlook.com)
Li, Jieyu (lijieyu@chinamobile.com)
Lv, Lily (lvyunping@huawei.com)
Sivakolundu, Ramesh (sramesh@cisco.com)



Data center backend networks are the primary use case for SFC

- A data center backend network is a crucial part of the data center infrastructure, specifically designed **to handle high-bandwidth, low-latency communication between high-performance computing resources like GPUs, storage, and other critical components within the data center. It's optimized for AI training and inference (AI) workloads, as well as other high-performance computing (HPC) tasks**, ensuring efficient data transfer and processing.

> SFC basic concepts and comparisons to other tools (PFC, QCN, CI, ...):

- <https://www.ieee802.org/1/files/public/docs2022/new-congdon-SFC-proposal-0322-v01.pdf> summarizes background, SFC concepts, and comparisons to QCN and CI.
- <https://www.ieee802.org/1/files/public/docs2022/new-bottorff-sfc-0322-v6.pdf> describes the use cases of SFC and its comparison with DCQCN and PFC.
- <https://www.ieee802.org/1/files/public/docs2025/dw-chen-sfc-computation-simulation.pdf> explains SFC remains highly useful with AI workloads.

> SFC design, computation and simulations:

- <https://www.ieee802.org/1/files/public/docs2022/new-blendin-SFC-sim-0522-v01.pdf> and <https://www.ieee802.org/1/files/public/docs2022/new-blendin-SFC-Simulation-Results-0722-v01.pdf> show the benefits of SFC and SFC w/ Proxy compared to PFC based on HPC workloads by simulation.
- <https://www.ieee802.org/1/files/public/docs2022/new-congdon-SFC-design-topics-0622-v01.pdf> gives 11 design topics.
- <https://www.ieee802.org/1/files/public/docs2025/dw-chen-sfc-proposal-0125-v01.pdf> provides proposals on drafting the text and updated the individual text <https://www.ieee802.org/1/files/public/docs2025/dw-chen-individual-text-0325-v03.pdf> based on previous contributions.
- <https://www.ieee802.org/1/files/public/docs2025/dw-chen-sfc-computation-simulation.pdf> describes how to compute SFC parameters and simulations based on AI workloads.

> Note: not all previous contributions are listed here.

3 types of congestions that may trigger flow control

- Incast congestion
 - > PFC: congestion spreading.
 - > SFC: preferred solution.
- In-network congestion
 - > Load balancing (packet spray, adaptive routing, etc.): preferred solution.
 - > ECN: no lossless guarantee.
 - > SFC: an option.
- Outcast congestion
 - > PFC: preferred solution.
 - > SFC: equal to PFC.
- Packet trimming BTS (back-to-sender):
 - > No lossless guarantee, triggered when packets are dropped due to buffer overflow.

SFC goals and Clarifications

- For lossless networks
 - > First SFC to avoid PFC spreading
 - > Then PFC for lossless
- For best effort networks
 - > First SFC to avoid PFC spreading
 - > Then packet trimming for fast retransmission

Notes:

1. PFC spreading can be a severe issue as the radix (number of ports per switch) is very high in data center backend networks.
2. AIDC workloads are highly dynamic, characterized by ON/OFF traffic patterns and pervasive incast.
3. Packet trimming BTS (back-to-sender) is a drop notification that the dropped packet is trimmed and directly sent back to the sender.

Draft status and Proposal for the next step

- The latest individual text and its brief can be found at <https://www.ieee802.org/1/files/public/docs2025/dw-chen-individual-text-0325-v03.pdf> and <https://www.ieee802.org/1/files/public/docs2025/dw-chen-text-status-and-todos.pdf>
- The introducing (clause 1-6) and the concept and component description (52.1-52.4) parts are almost there.
- The management part (12, 48) can be handled last.
- Clause 52.5 is the meat. For **next step**:
 - > Update and finalize a first but complete version of SFCP procedure (52.5.2).
 - > Update Encoding (52.5.3) based on what has been proposed in this contribution.
 - > Revise Variables (52.5.1) accordingly.
 - > Add Buffer requirements for SFC (Annex Y) based on the calculation given in <https://www.ieee802.org/1/files/public/docs2025/dw-chen-sfc-computation-simulation.pdf>
- Then, **take the draft to a Task Group ballot**. (Sep.2025?)

Discussion

An example to feel the incast in AIDC

During the process of answering my question, how many times did the incast of combine all to all communication occur, considering prefill and decode?

- DeepSeek answering my question:
 - > M=492 (492 output tokens for this answer), L=61 (DeepSeek V3/R1)
 - > **Causing more than 30,000 seven-to-one incast**, considering 8 experts on 8 GPUs are activated.
 - > Typical data size for one incast ranging from hundreds of KB to hundreds of MB.
- The AI computing modules (GPUs) used for inference are interconnected through the data center's backend network, with deployments typically ranging from several hundred to several thousand units.
 - > - Scalable Task Distribution: The system seamlessly scales from a single query to thousands, dispatching each task across available GPUs.
 - > - Layered Expert Coordination: Each computational layer activates a group of 8 experts: one dispatches data to the other seven, then aggregates their outputs (incast). Certain "hot experts" may receive a higher concentration of tasks.
 - > - Rapid Execution: Each layer completes on the order of 100 microseconds, enabling fast, efficient responses.
- **Incast is constantly happening, with an extremely large and highly dynamic volume.**

- If you asked a question requiring M output tokens, and the model has L layers:
 - Total incasts $\approx L * (1 + M)$ (1 for prefill, M for decode steps).

During the process of answering my question, how many times did the incast of combine all to all communication occur, considering prefill and decode?

In the context of large language model inference (like the one you're interacting with), the terms "incast" and "all-to-all communication" typically arise in distributed computing scenarios, particularly when the model is sharded across multiple devices (e.g., GPUs or TPUs). These communication patterns are crucial for handling the dependencies between different layers or attention heads in transformer-based models.

Key Concepts:

1. **Incast:** A many-to-one communication pattern where multiple devices send data to a single device (common in aggregation steps).
2. **All-to-all:** A communication pattern where every device exchanges data with every other device (common in attention computations or tensor parallelism).
3. **Prefill Phase:** Processing the entire input prompt in parallel (requires matrix multiplications and attention computations).
4. **Decode Phase:** Generating tokens auto-regressively (one token at a time, with smaller but repeated communication steps).

Communication in Transformer Inference:

- During **prefill**, the model computes self-attention over the entire input sequence. This involves:
 - **All-to-all** for attention scores (e.g., query-key matmul, value aggregation).
 - Potential **incast** when aggregating results (e.g., reducing partial outputs from parallel heads or devices).
- During **decode**, the model generates one token at a time, but still requires:
 - **All-to-all** for incremental attention (though less intensive than prefill).
 - **Incast** when combining results across devices (e.g., logit aggregation).

Estimating Communication Occurrences:

The exact number of incast or all-to-all steps depends on:

- The model architecture (e.g., number of layers, attention heads, tensor/pipeline parallelism).
- The hardware setup (e.g., how the model is partitioned across devices).

For a typical large transformer model (like GPT-3 or similar):

1. **Prefill Phase:**
 - Each layer performs self-attention, which involves at least one all-to-all for attention computation.
 - Incast may occur during residual connections or layer normalization aggregation.
 - If the model uses tensor parallelism, each matmul may require an all-reduce (a form of incast).
2. **Decode Phase:**
 - Similar communication patterns, but repeated sequentially for each generated token.

Approximate Count:

Assuming:

- The model has L layers.
- Tensor parallelism is used (e.g., with H devices).
- Prefill processes M tokens, and decode generates N tokens.

Then:

- **Prefill:**
 - All-to-all: $O(L)$ per token, but parallelized \rightarrow roughly L all-to-all steps across devices.
 - Incast: $O(L)$ (e.g., for residual additions or all-reduces).
- **Decode:**
 - All-to-all: $L * H$ (per-token, per-layer).
 - Incast: $L * H$.

For This Specific Query:

Since I (the model) didn't explicitly detail my distributed setup, I can't give you an exact number. However, in a typical scenario:

- If you asked a question requiring M output tokens, and the model has L layers:
 - Total incasts $\approx L * (1 + M)$ (1 for prefill, M for decode steps).
 - Prefill all-to-all \approx similar order.

Example:

If the model has 80 layers and generates 10 tokens:

- Incasts $\approx 80 * (1 + 10) = 880$.
- All-to-all \approx similar.

Would you like a more hardware-specific breakdown (e.g., for TPU pods or GPU clusters)?

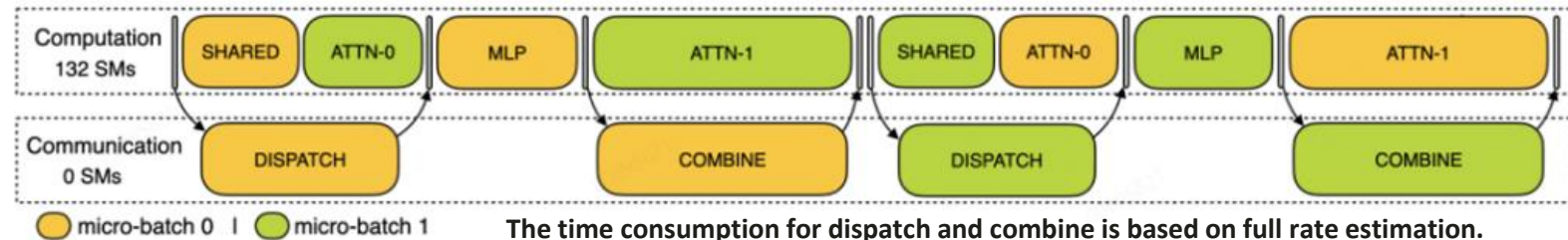
AIDC network is quite homogeneous

- **Networking (Topology, bandwidth, latency):**

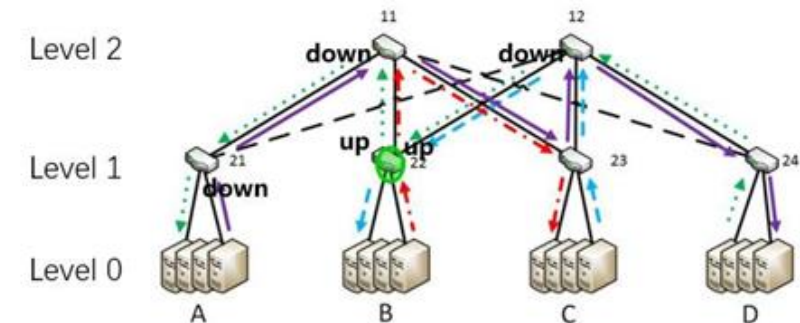
- > 2 or 3-layer CLOS/fat-tree is the main topology, interconnecting thousands or tens of thousands of GPUs.
- > High bandwidth (e.g., 400G ports), high radix (e.g., 128 ports per switch) and ultra-low latency (down to 250ns switch delay¹).

- **Collective communication:** Whether training or inference and regardless of parallelism method, all jobs rely on collective operations like AllReduce and AllToAll.

- **Overlap design:** Maximize the overlap between computation time and communication time to optimize Job Completion Time (JCT).



- Therefore, simulation and verification cases in <https://www.ieee802.org/1/files/public/docs2025/dw-chen-sfc-computation-simulation.pdf> employ representative 2-layer CLOS combined with AllReduce and AlltoAll workloads.
- ***Capital intensive:** In 2024, global DC capital expenditure (CAPEX) reached \$450B due to a surge in AI computing power demand, with network infrastructure (including high-speed interconnects, optical modules, switching equipment, etc.) accounting for approximately 15-20%.



<https://mentor.ieee.org/802.1/dcn/24/1-24-0028-04-lCne-aicn-report-draft.pdf>

SFC Design Principles: easy to use

- Ideally, SFC should be almost as simple and easy to use as PFC.
- Step 1: preconfigure SFC threshold and pause time (informative on the perspective of SFC standard)
 - > Option a) Know your topology, switch capabilities, and the AI application you run (relates to the level of incast), then calculate as in <https://www.ieee802.org/1/files/public/docs2025/dw-chen-sfc-computation-simulation.pdf>
 - > Option b) Using empirical values summarized from typical scenarios
- Step 2: enable SFC on switches. Enable proxy mode if end-stations do not support SFC.
- NOTE: SFC trigger-to-activation latency is very low with no significant jitter in data center backend networks.
 - > Max. 2 hops (2-layer CLOS) or 4 hops (3-layer CLOS).
 - > Switch (fixed) delay down to 250ns per hop, queuing delay for SFCM at high priority is negligible.
 - > Plus the processing time related to SFC procedures.

SFCM format design

- General idea: Design a unified format that can be used in both AI and HPC data centers.
- What is needed (mandatory):
 - > Source address of the congesting frame as SFCM's destination address
 - > Indication of the class to be paused as well as the pause time.
- Encapsulation options: SFCM PDU in Ethernet (EtherType 0x89A2) or UDP IPv4 or UDP IPv6.
 - > Including the address.
- SFCM PDU: Version + Reserved + TLV. For example,
 - > Value of TLV Type A: referencing the PFC format in a standard PFC frame that includes pause time associated with the class.
 - > Value of TLV Type B: some first bytes of the congestion frame's MSDU and an explicit pause time.