Host-INT* for packet-telemetry

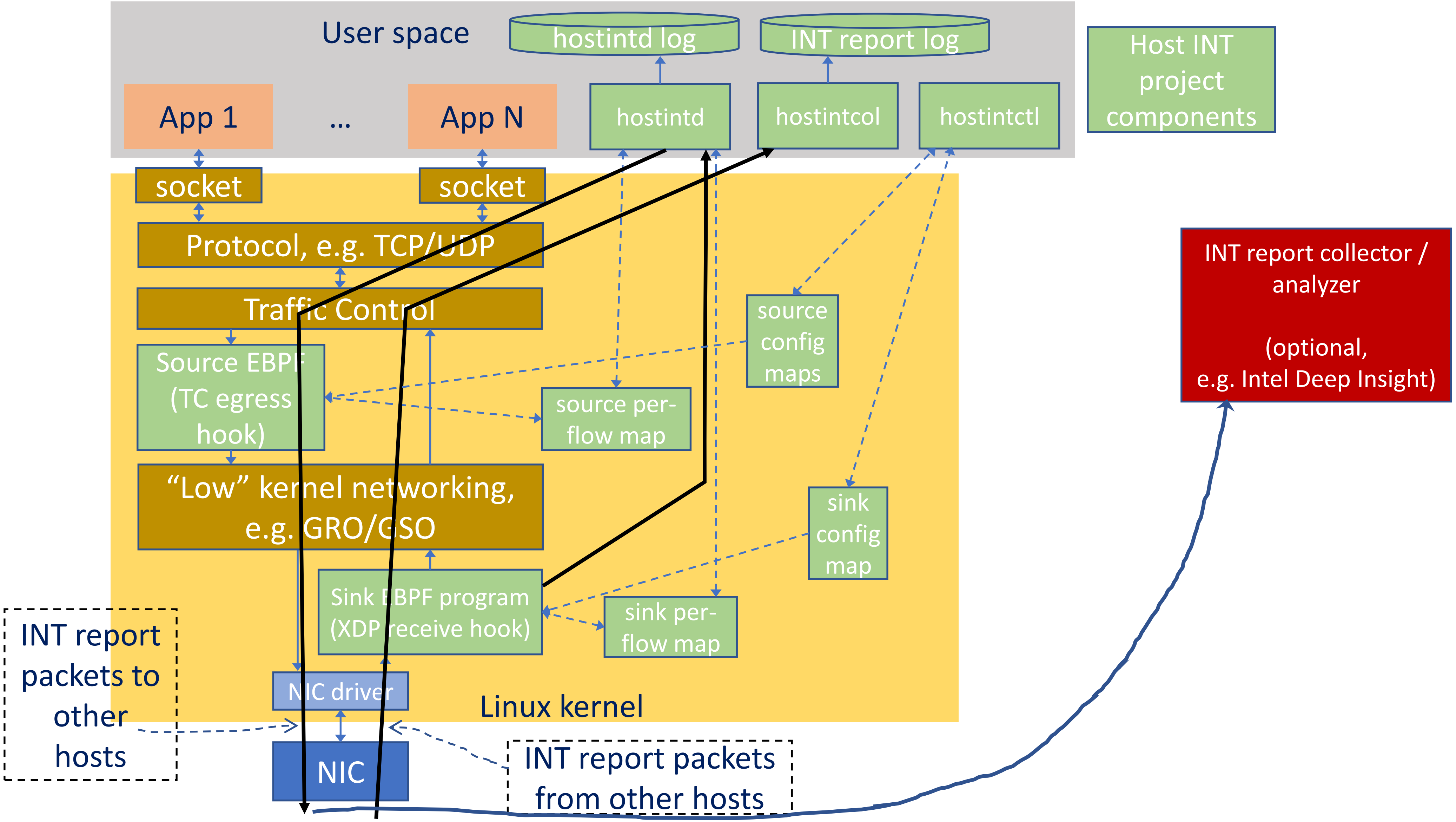
Intel Corporation

2021-Jun-30

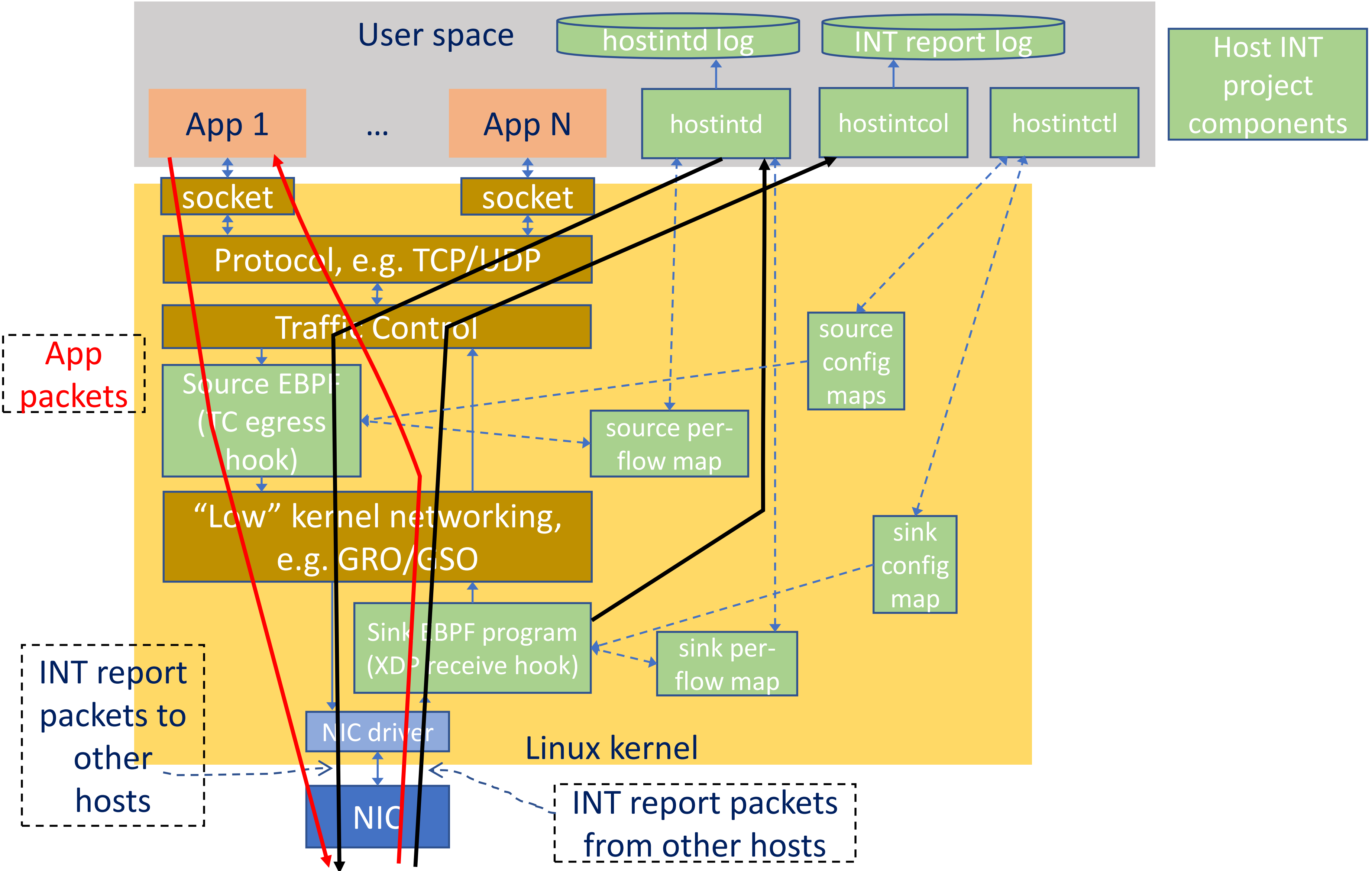
* Other names and brands may be claimed as the property of others.

- The full name of this project is “Host-INT for packet-telemetry”
  - We will usually refer to it as Host-INT in conversation.

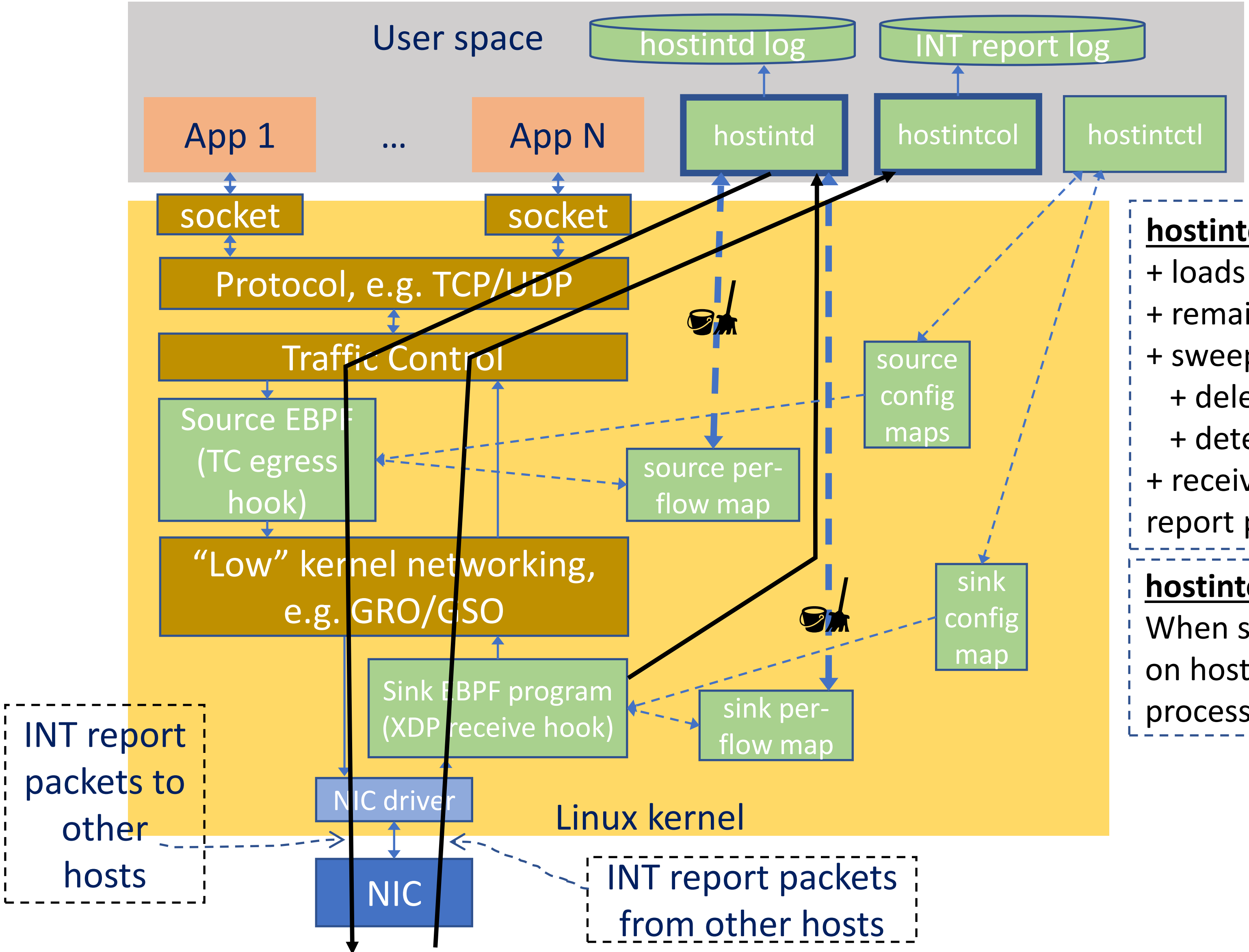
Host-INT project structure (with TC egress source)


User space
hostintd log
INT report log
Host INT project components
App 1
…
App N
hostintd
hostintcol
hostintctl
socket
socket
Protocol, e.g. TCP/UDP
Traffic Control
Source EBPF (TC egress hook)
source config maps
source per-flow map
INT report collector / analyzer
(optional, e.g. Intel Deep Insight)
"Low" kernel networking, e.g. GRO/GSO
sink config map
Sink EBPF program (XDP receive hook)
sink per-flow map
INT report packets to other hosts
NIC driver
Linux kernel
NIC
INT report packets from other hosts

Host INT project structure



hostintd, hostintcol



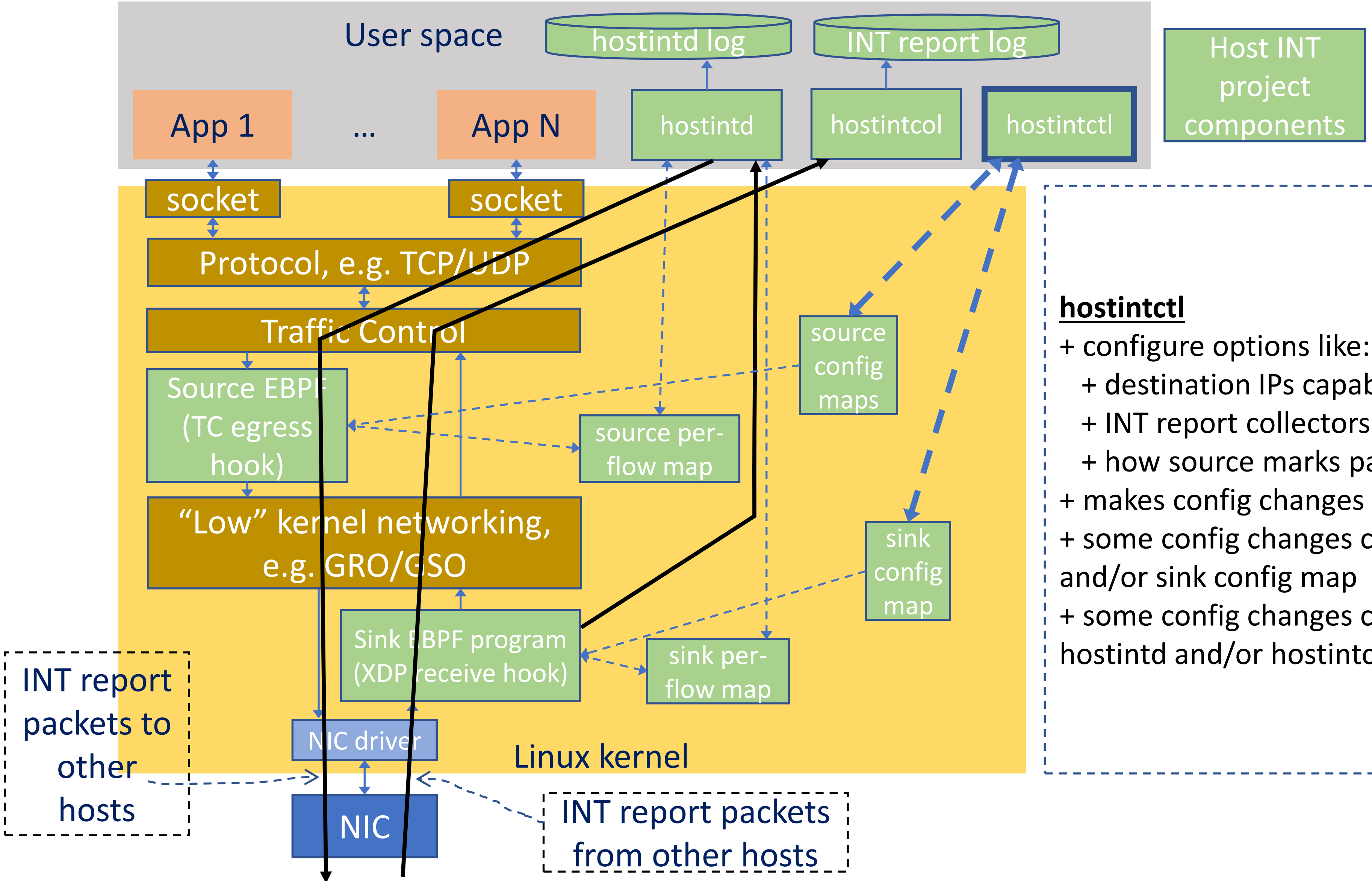
hostintd

+ loads EBPF programs into kernel
+ remains running while EBPF programs are running
+ sweeps over EBPF per-flow maps to:
  + delete stale flow entries
  + detect packet losses and generate INT drop reports
+ receives copied packets from sink and sends INT latency report packets to configured report collectors

hostintcol

When source is running on host A, hostintcol is also running on host A, and receiving INT report packets from hostintd process of hosts to which A is sending.

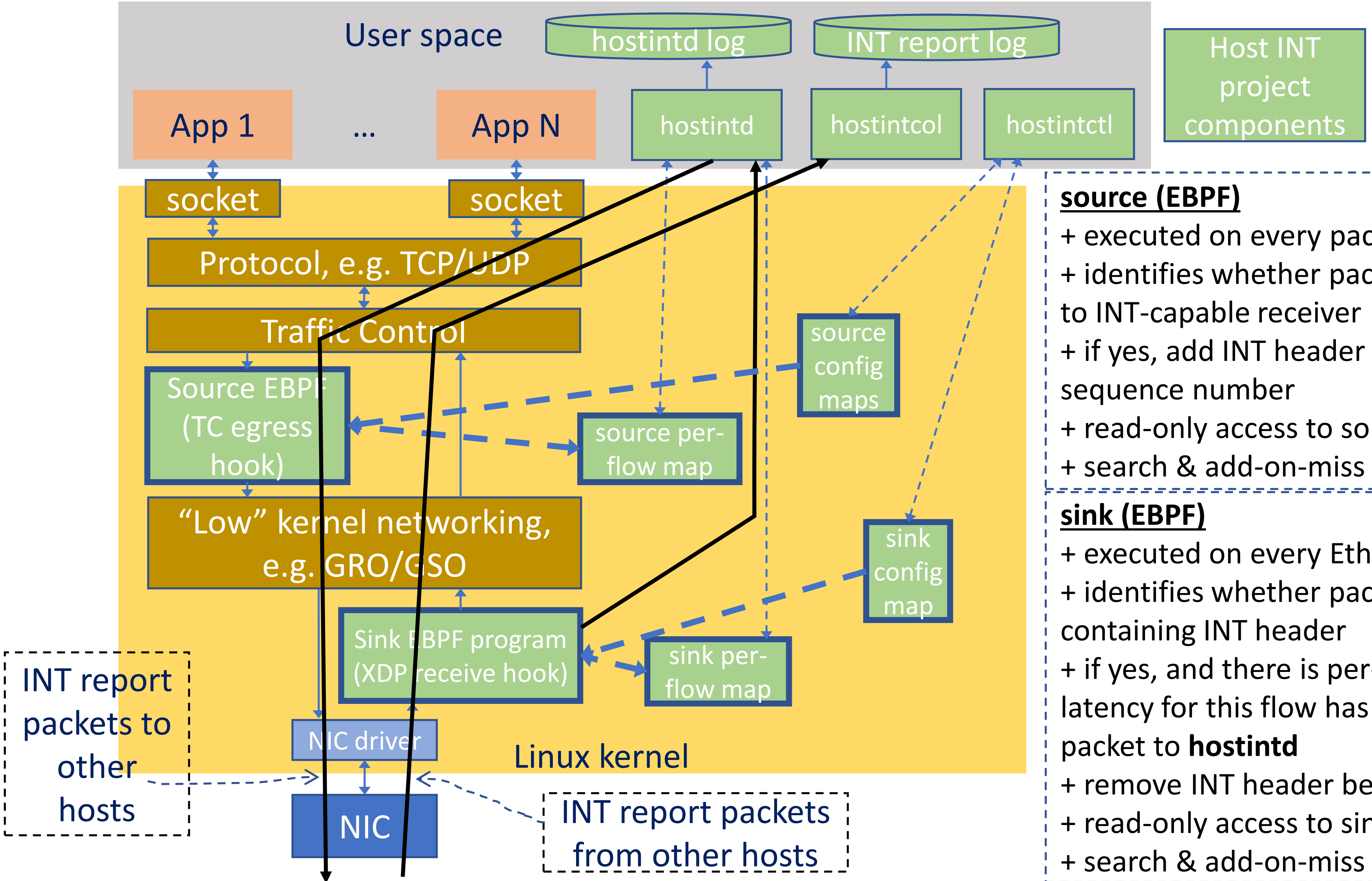
hostintctl



hostintctl

+ configure options like:
  + destination IPs capable of receiving INT packets
  + INT report collectors to which reports are sent
  + how source marks packets with INT headers
+ makes config changes quickly, then exits
+ some config changes cause modifications to source and/or sink config map
+ some config changes cause changes to config used by hostintd and/or hostintcol

source & sink EBPF programs



source (EBPF)

+ executed on every packet from application

+ identifies whether packet is IPv4+TCP/UDP and destined to INT-capable receiver

+ if yes, add INT header containing timestamp and per-flow sequence number

+ read-only access to source config map

+ search & add-on-miss access to source per-flow map

sink (EBPF)

+ executed on every Ethernet frame from NIC

+ identifies whether packet is IPv4+TCP/UDP and marked as containing INT header

+ if yes, and there is per-flow sequence number gap, or latency for this flow has changed significantly, send copy of packet to **hostintd**

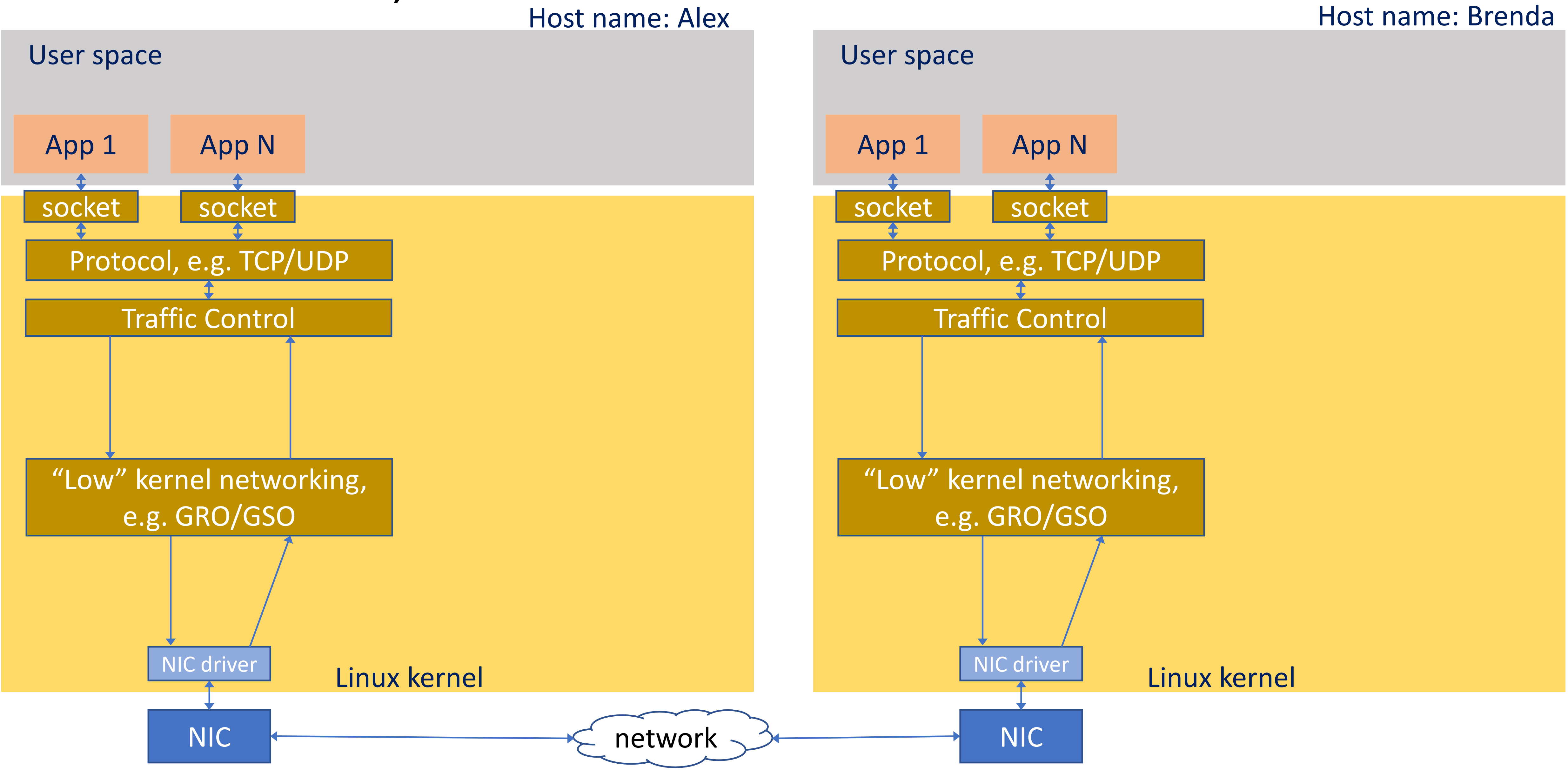
+ remove INT header before sending packet to kernel

+ read-only access to sink config map

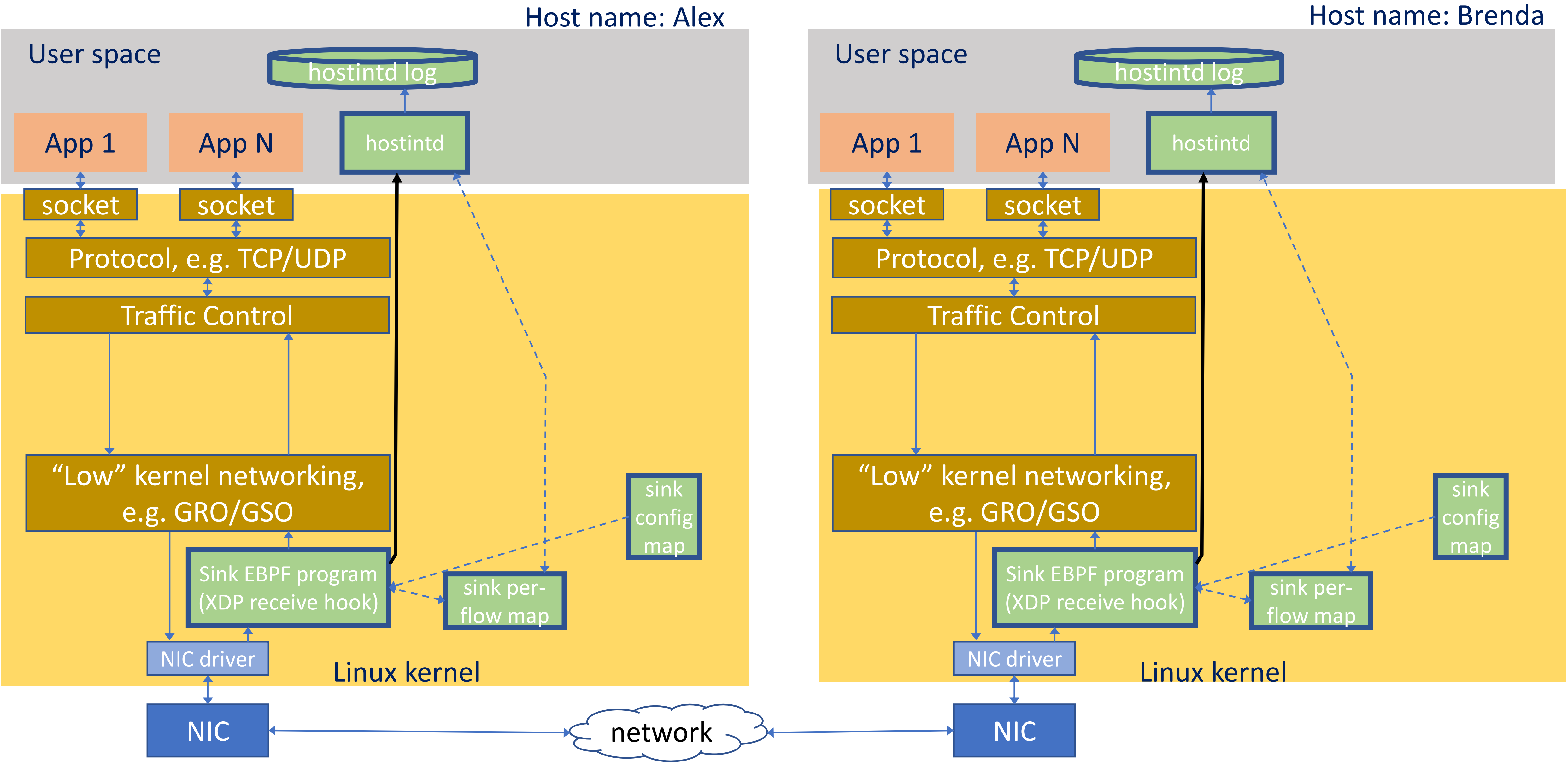
+ search & add-on-miss access to sink per-flow map

Example event sequence

Boot time, before Host INT software started



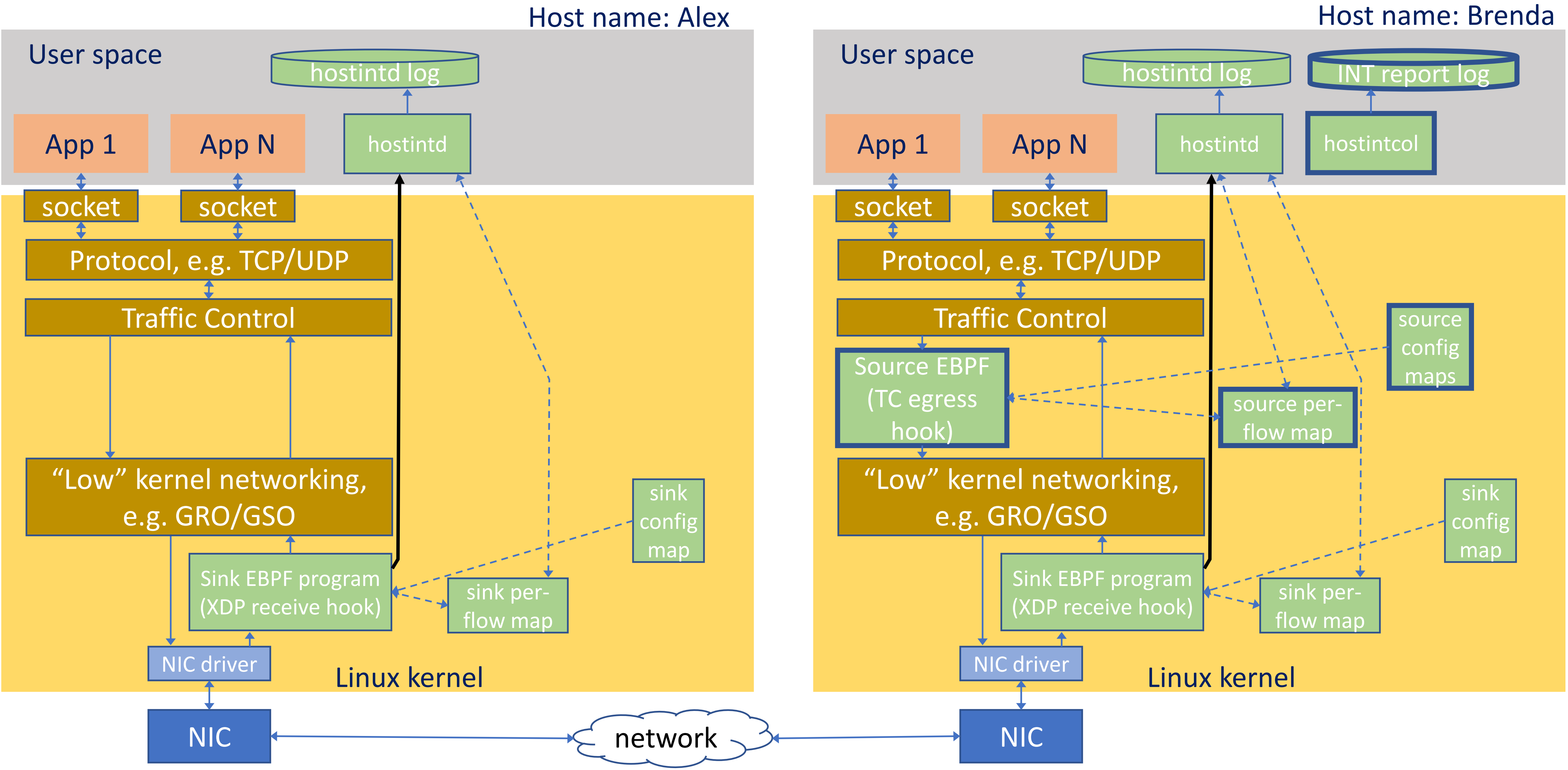
After hostintd started & loads sink EBPF



Notes on state after only sink is loaded

- Assumption:
    - In this example network, there is no hardware or software that will ever add INT headers to packets, except Host INT software
- Since no hosts have loaded source EBPF programs
    - No packets will have INT headers added to them
- For any received packet without INT header, the sink EBPF program will only:
    - parse packets up to IPv4 header
    - determine from DSCP field that packets do not have INT header
    - Pass the packet unmodified to the Linux kernel
    - No reports will be generated
    - Should be very quick and light on CPU resources
- But all hosts are now <u>ready</u> to receive packets with INT headers

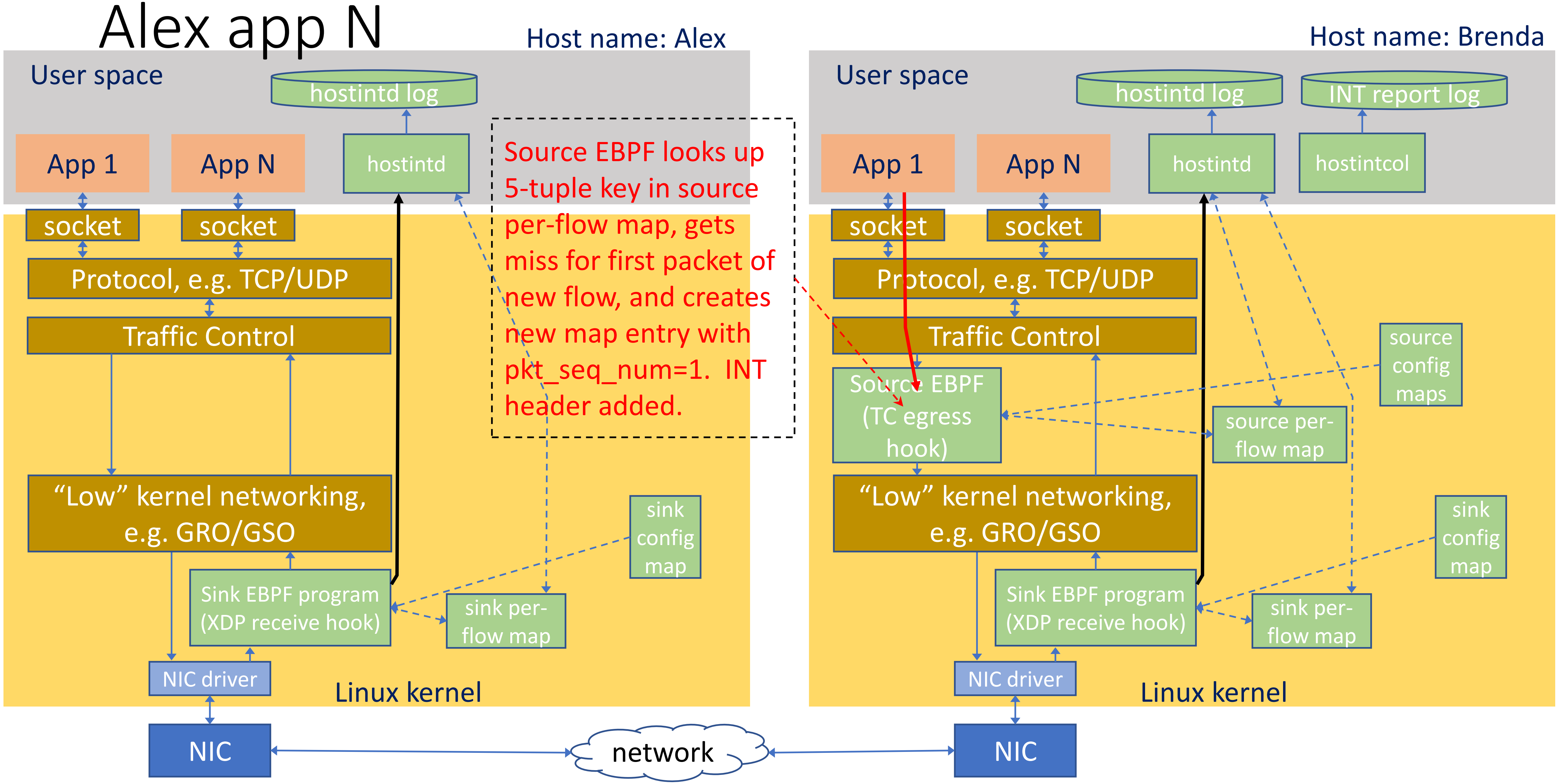
After source loaded on host Brenda



Notes on state after source loaded on Brenda

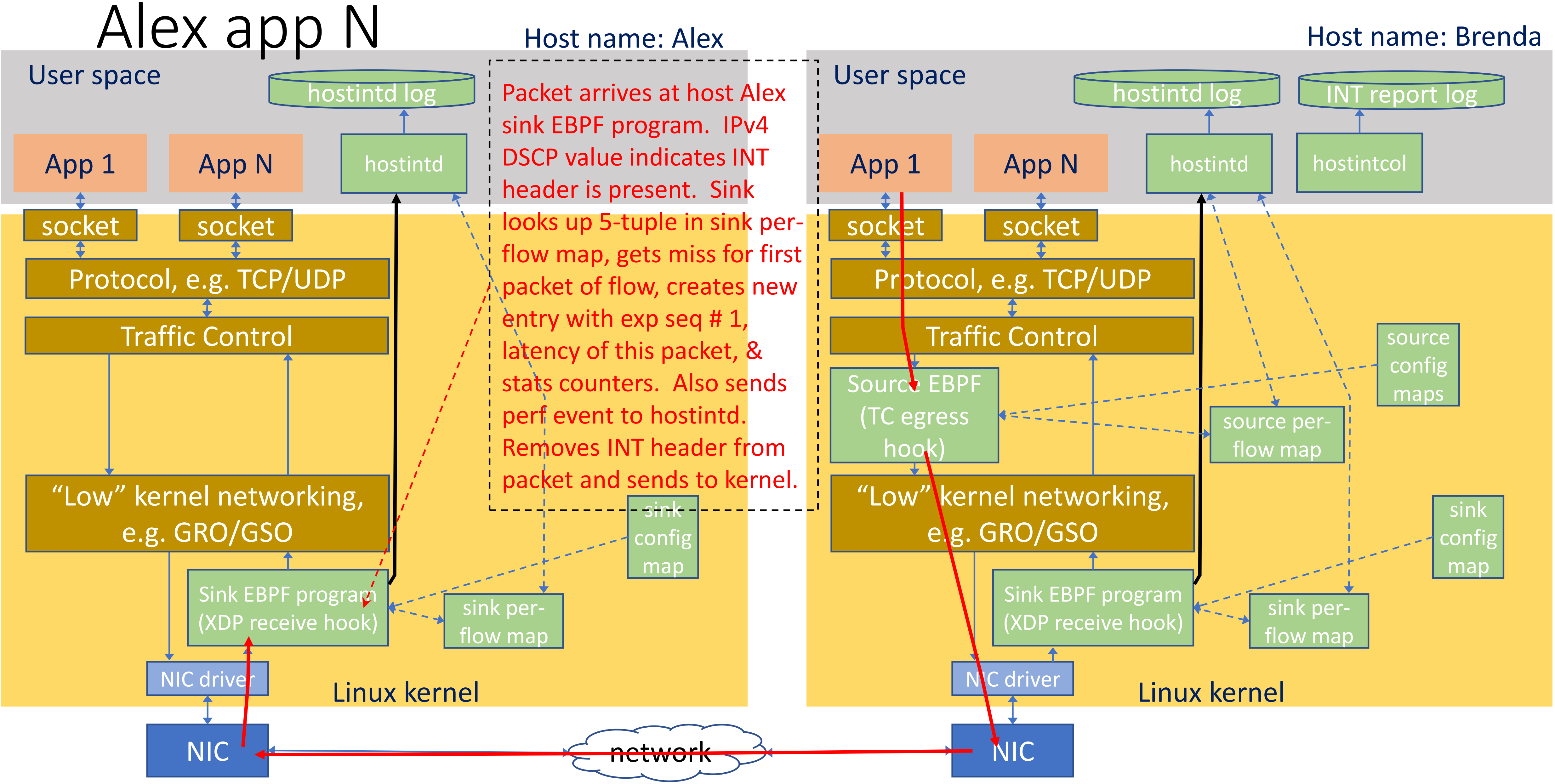
- In this example, we have configured all hosts to send INT reports back to the sender of the packet that caused the report to be generated.
- Thus no central collector of all INT reports
    - Host INT does support sending INT reports to both the sender of the packet, AND one or more central INT report collectors

Host Brenda app 1 sends TCP/UDP packet to Host Alex app N



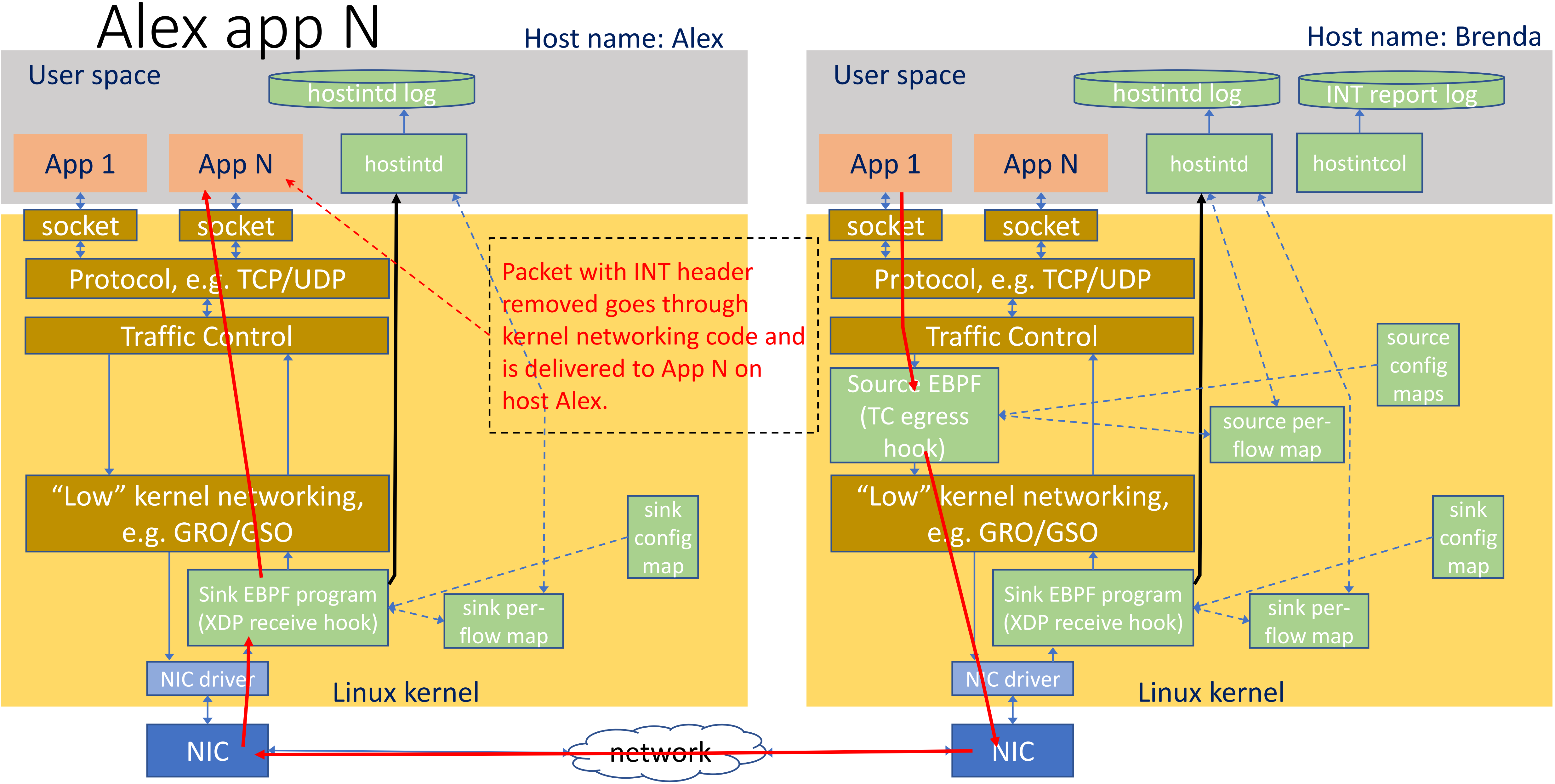
Source EBPF looks up 5-tuple key in source per-flow map, gets miss for first packet of new flow, and creates new map entry with pkt_seq_num=1. INT header added.

Host Brenda app 1 sends TCP/UDP packet to Host Alex app N



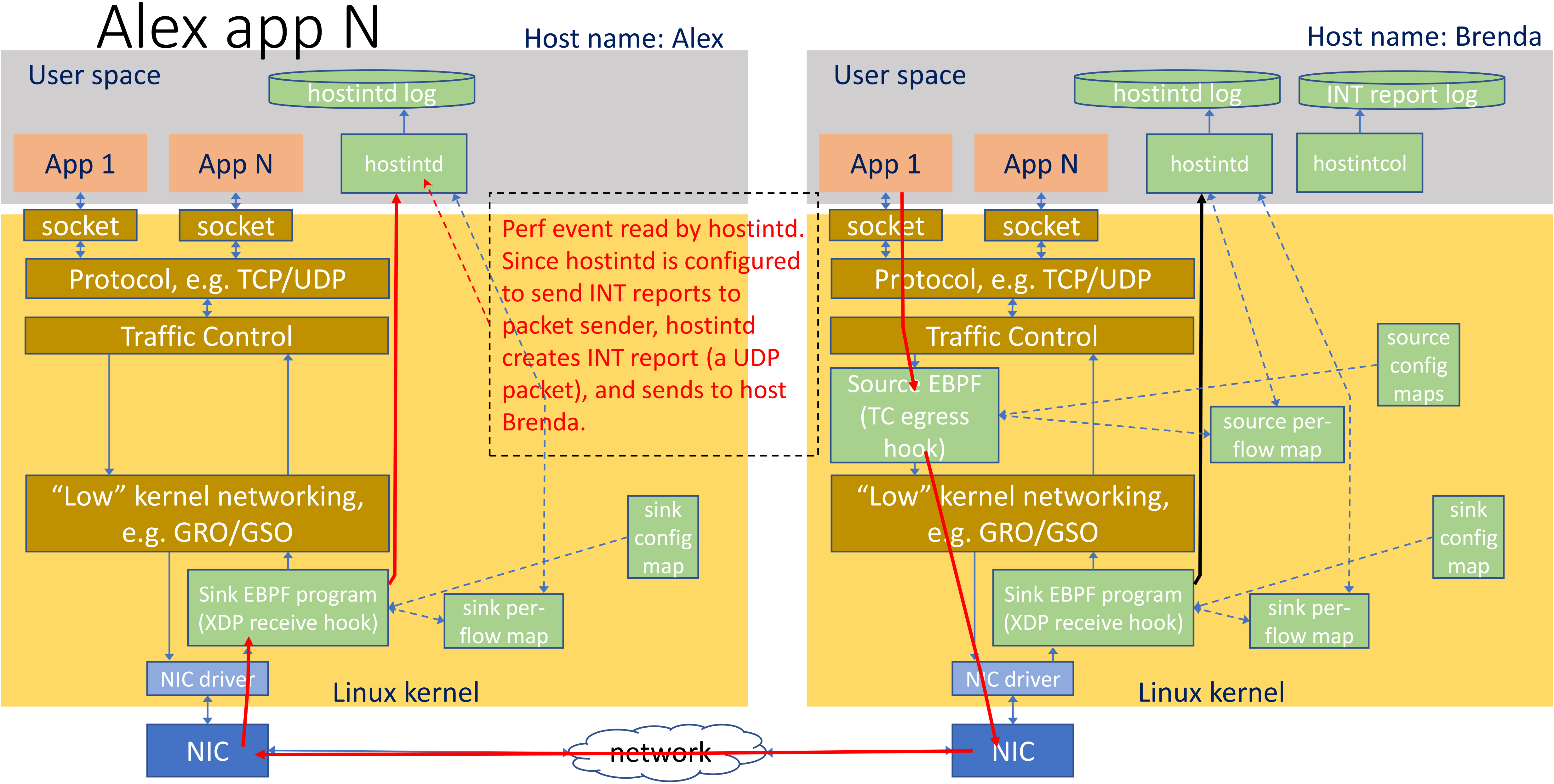
Packet arrives at host Alex sink EBPF program. IPv4 DSCP value indicates INT header is present. Sink looks up 5-tuple in sink per-flow map, gets miss for first packet of flow, creates new entry with exp seq # 1, latency of this packet, & stats counters. Also sends perf event to hostintd. Removes INT header from packet and sends to kernel.

Host Brenda app 1 sends TCP/UDP packet to Host Alex app N



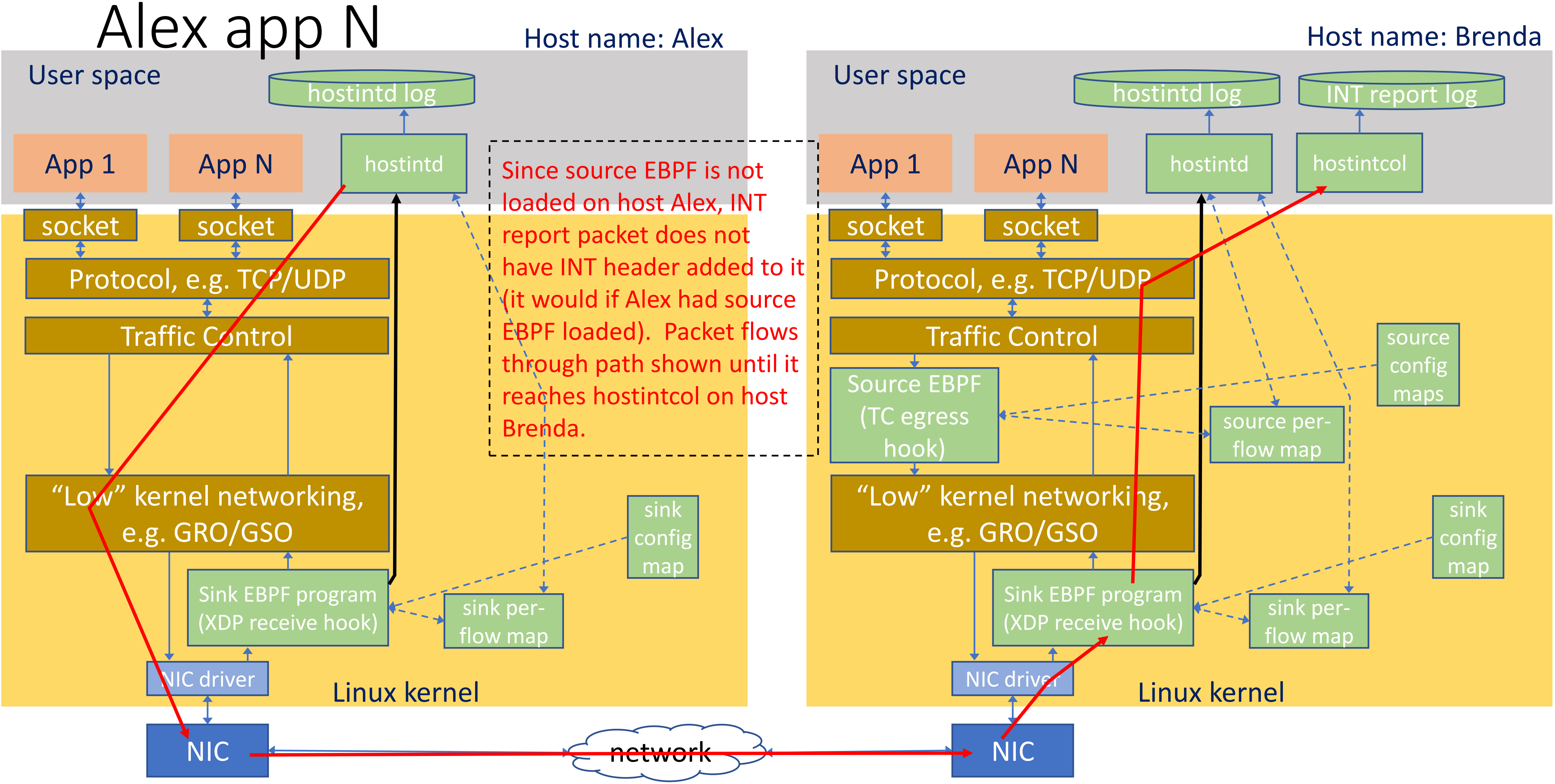
Packet with INT header removed goes through kernel networking code and is delivered to App N on host Alex.

Host Brenda app 1 sends TCP/UDP packet to Host Alex app N



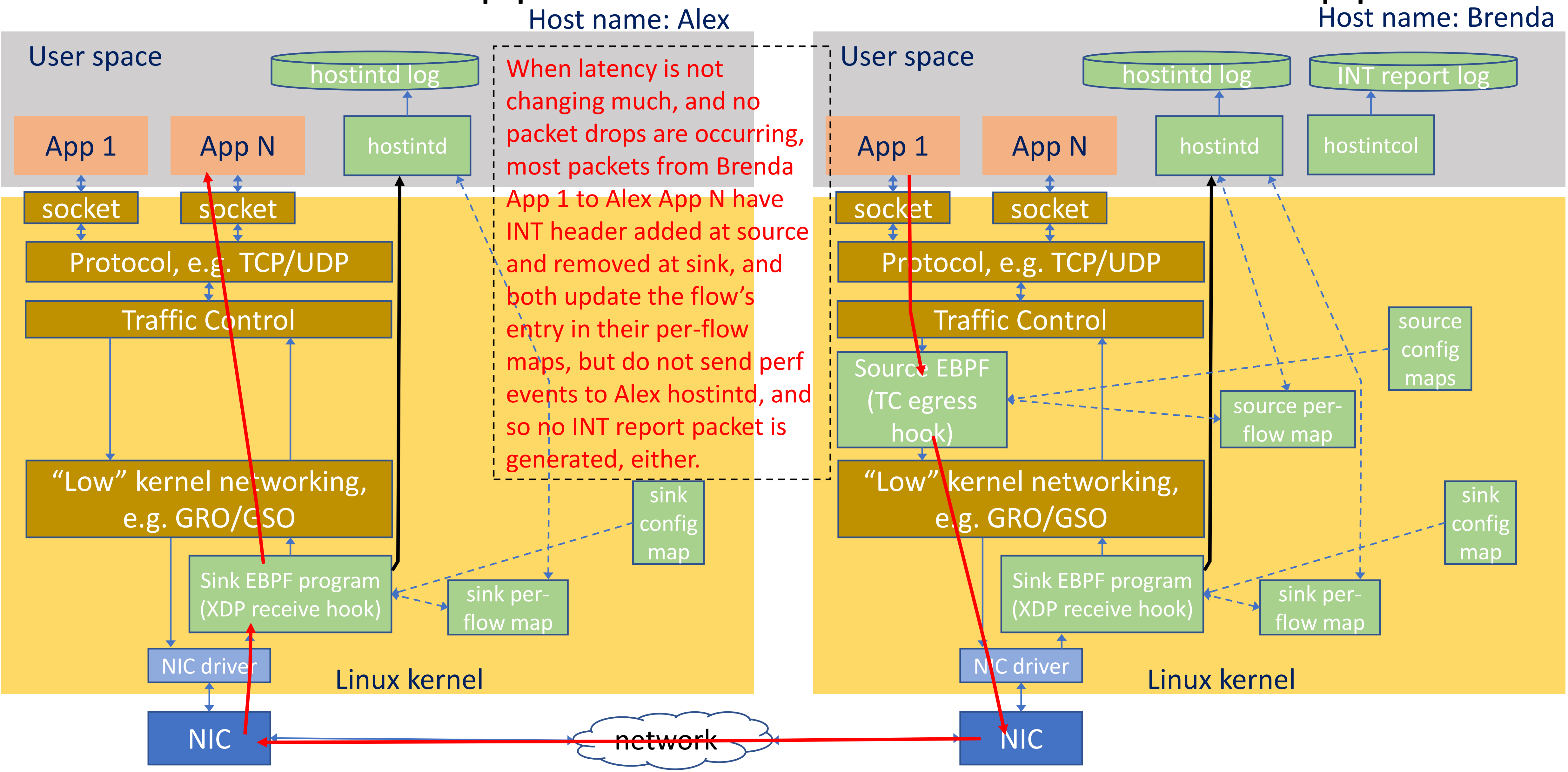
Perf event read by hostintd. Since hostintd is configured to send INT reports to packet sender, hostintd creates INT report (a UDP packet), and sends to host Brenda.

Host Brenda app 1 sends TCP/UDP packet to Host Alex app N



Since source EBPF is not loaded on host Alex, INT report packet does not have INT header added to it (it would if Alex had source EBPF loaded). Packet flows through path shown until it reaches hostintcol on host Brenda.

Host Brenda app 1 sends more to Host Alex app N



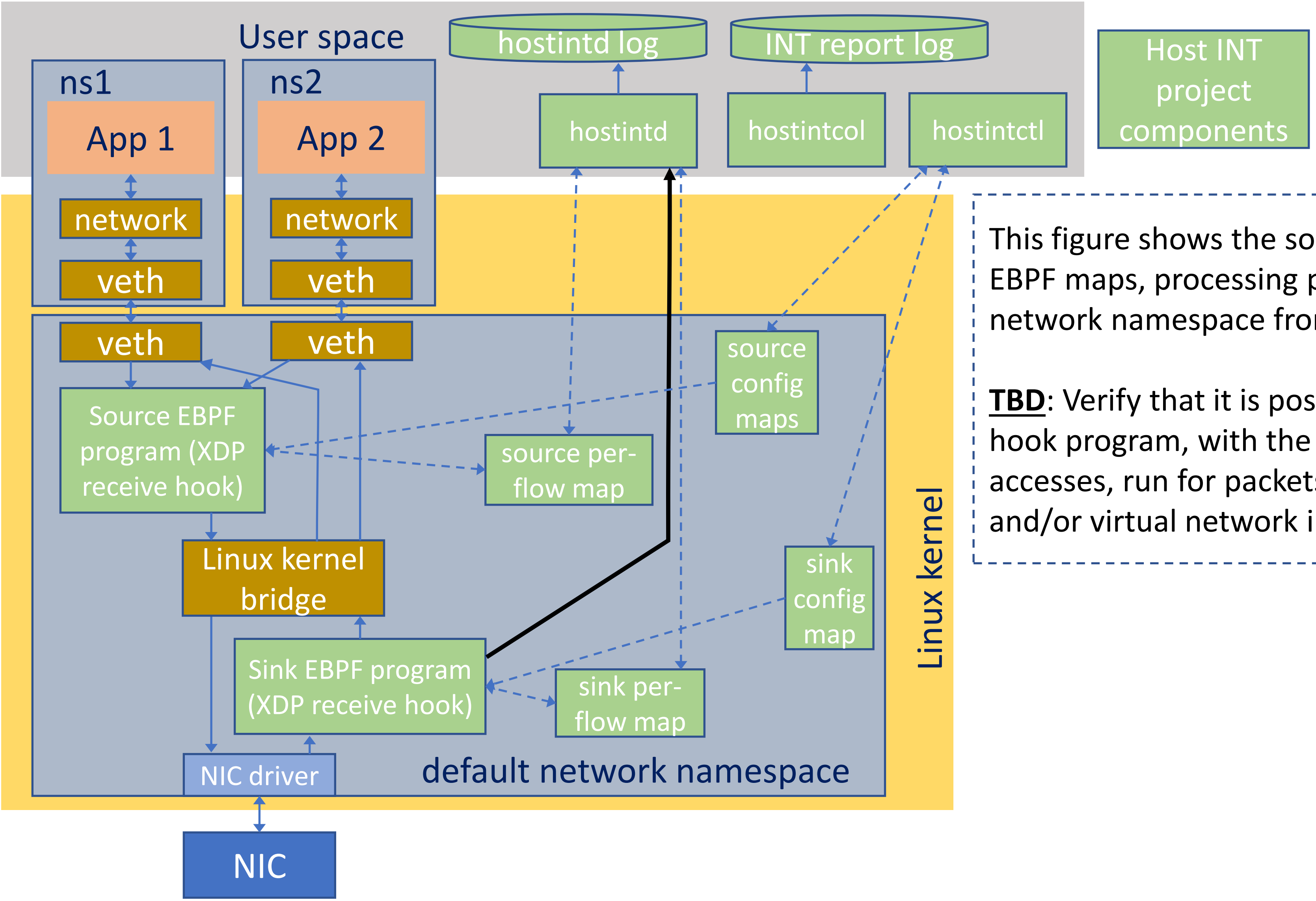
When latency is not changing much, and no packet drops are occurring, most packets from Brenda App 1 to Alex App N have INT header added at source and removed at sink, and both update the flow's entry in their per-flow maps, but do not send perf events to Alex hostintd, and so no INT report packet is generated, either.

XDP source program scenarios

XDP source program

- Previous figures have shown a deployment scenario using the TC egress hook EBPF program for adding INT headers at source host
    - Both the source (on TC egress hook) and sink (on XDP receive hook) EBPF programs can be loaded into kernel within the default Linux kernel network namespace
    - TBD: does it work if they are installed within a non-default network namespace?
- There may be some deployments involving containers and/or non-default network namespaces where it would be useful to instead:
    - Load an XDP receive hook EBPF program on a veth interface that adds INT headers at the source host
    - The next figure is one example of this, using a Linux kernel bridge to forward packets between network interfaces (some physical, some virtual) in the default network namespace.
    - Many other arrangements are possible.

Host INT project structure (with XDP source)



This figure shows the source EBPF program, with one set of EBPF maps, processing packets received in the default network namespace from two veth interfaces.

TBD: Verify that it is possible to have a single XDP receive hook program, with the same set of EBPF maps that it accesses, run for packets received from multiple physical and/or virtual network interfaces.