

# Stream Handlers: Application -Specific Message Services on Attached Network Processors

Ada Gavrilovska, Kenneth Mackenzie, Karsten Schwan,  
and Austen McDonald

Center for Experimental Research in Computer Science  
Georgia Tech

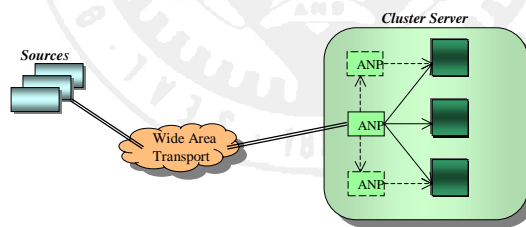


## Problem description

- Increasing number of services are supported on programmable network processors
  - software routing, packet classification, network monitoring, load balancing...
  - existing service operate on receive-side, mostly on packet TCP/IP headers
- Streaming applications can benefit from NI-based stream specialization
  - need to look 'beyond' packet header into through application-level content

## ANP-resident services

- Examples of stream specializations in a cluster attached network processor (ANP):
  - filtering of sensor data or large data in scientific applications
  - content-based stream differentiation or multicast
  - scheduling in multimedia applications
  - execution of application specific mirroring codes for large scale OIS
- ANP acts as a stream pre-processing engine for a set of cluster nodes

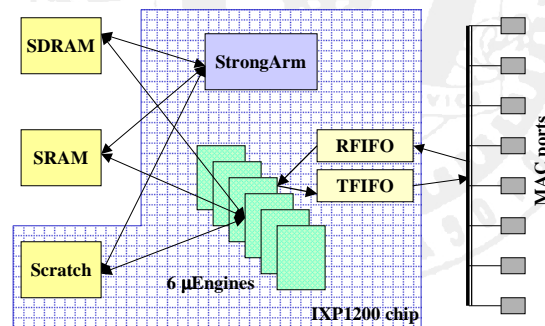


## Goal

- Enrich packet movement in the ANP to execute application specific services at link speeds
- Enable stream processing beyond packet header
- Key insight:
  - stream handlers
- Implementation vehicle:
  - Intel's IXP1200

## Intel's IXP1200

- o IXP1200 – programmable hardware optimized for movement of large volumes of network packets between its ports
- o Data received as an array of 64B MAC-layer packets



## Overview

- o Motivation and approach
- o Stream handlers
  - receive side and transmit side processing
- o Experimental evaluation
- o Conclusion and future work

## Approach: stream handlers

- Provide efficient implementation of lightweight computational units – **stream handlers** – to perform stream processing at link speeds
- Stream handlers executed on ANP in ‘headroom’ available on packets’ fast path at receive-, transmit-side or both
- Variable handler complexity can be sustained at link speeds depending on application-level message sizes

## Accessing application-level data

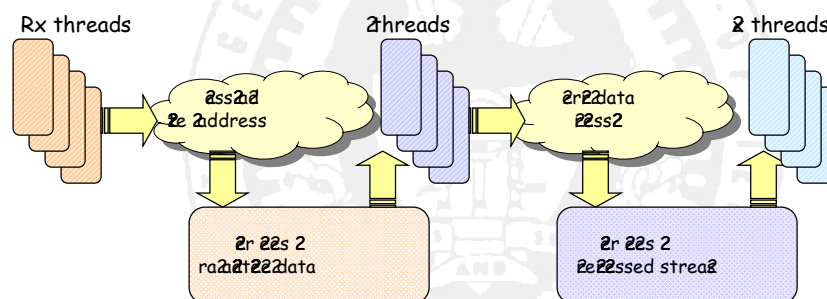
- Stream processing actions manipulate both message headers and application-level content
- For arbitrary message sizes: implement reliable packet fragmentation and reassembly protocol (RUDP) to compose application level data units in IXP memory
- For arbitrary message content: understand application-level data structures using lightweight format descriptions (PBIO)

## Software architecture

- Initialization, management, and exception handling is controlled by StrongArm
- Microengine threads execute designated tasks: Rx, X, and Tx
- Rx and Tx threads perform packet reassembly and fragmentation; X execute handler

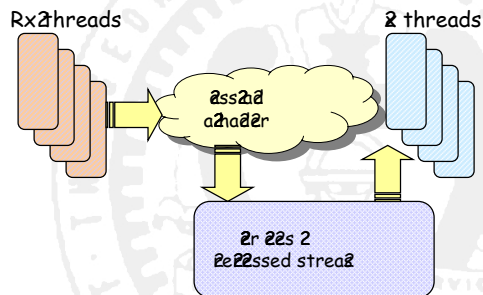
Rx => Mm => X => Mm => Tx

## Data plane operation



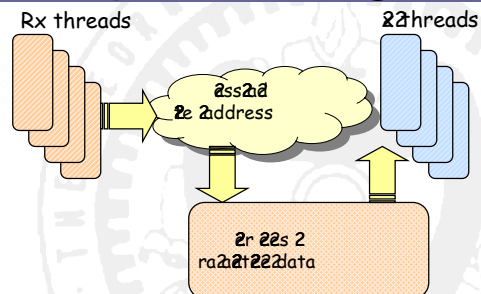
- Handler execution can be associated with ANP's receive side, transmit side or both

## Rx-side processing



- Receive side handlers - RxX threads
  - reduce unnecessary data movement for filtering applications
  - useful when amount of data 'touched' by the handler, and the state maintained across packets is small
  - existing header-only packet handlers are Rx-side only

## Tx-side processing



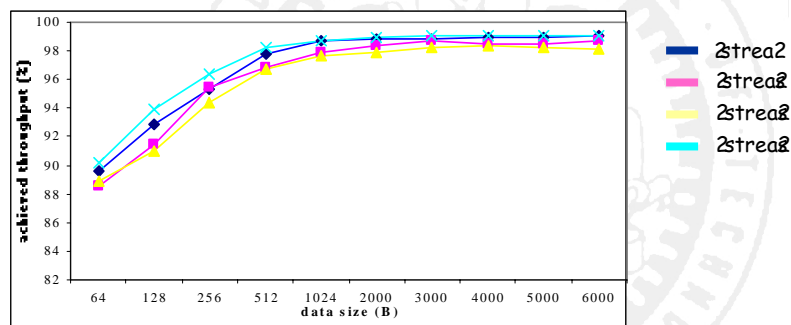
- Transmit-side TxX threads
  - reduces extra Mm copy when data can be modified as it's being sent out
  - transmit code less complex than receive code, therefore more complex handler code can be supported at line speeds
  - stream differentiation can be customized based on different clients

## Experimental evaluation

- Experiments conducted using SDK2.0 simulation package and Radisys ENP2505 boards
- Base performance of assembling application-level data is measured
- Effects of stream handlers of different complexity are evaluated
  - simple handler - f1 - 'touches' only first integer field in each data item (first MAC level packet)
  - complex handler - f2 - inspects every integer in data item

## Composing application-level data in the IXP is viable

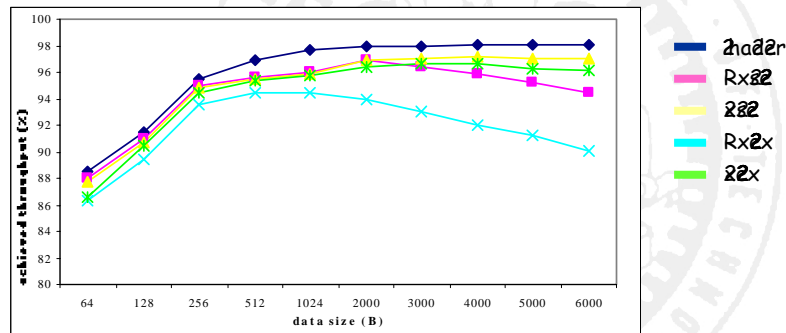
base performance without any handlers



- overheads amortized with larger data which spans multiple MAC packets
- parallelism hides overheads when number of streams increases

## Both Rx- & Tx-side handlers necessary

-and side handler execution without any filtering-



- Rx-side processing degrades performance as data sizes and complexity of handlers increase
- Tx-side processing achieves more constant performance levels

## Both Rx- & Tx-side handlers necessary

-and side execution of filtering-handlers

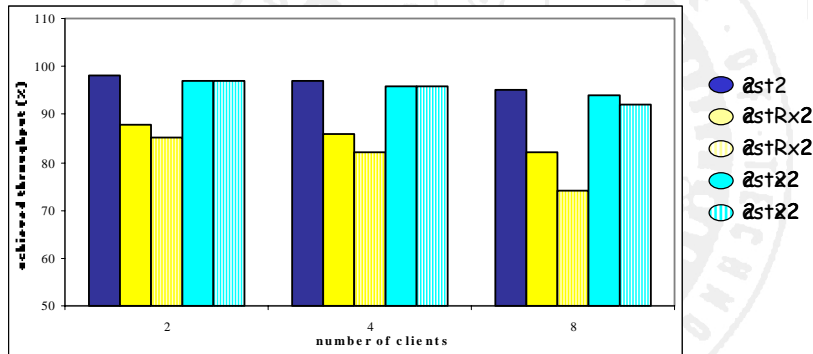
\	null	f1(Tx)	f1(Rx)	f2(Tx)	f2(Rx)
64B	91.03	90.96	93.66	88.12	86.66
2048B	98.55	98.45	98.98	94.02	84.14

↑  
rates & data

- Rx-side filtering improves performance for simple filters by reducing memory accesses
- As complexity of the filtering decision increases, Rx-side filtering becomes a bottleneck, and Tx-side filtering is a better solution

## Importance of Tx-side processing

multicast customized on per-client basis



- With Tx-side processing multicast customized on per-client basis can be achieved with small performance penalty

## Conclusion

- Use of lightweight stream handlers to perform application-specific stream specializations in the ANP
- Combination of receive- and transmit-side processing offers possibilities for richer set of services to be supported on the NI
- Efficient solution to rebuild application level data in ANP, and perform stream processing based on true application content

## Future Work

- Continue evaluating and testing the design
- Evaluate more complex thread scheduling mechanisms and assignments of tasks to Rx, Tx, and X threads
- Towards automated handler generation formalize relationships between stream data units, handler complexity and IXP resource management
- Automate format and handler placement onto ANP through an API for 'programming' the ANP

Thank you!

