Cross Layer Feedback in Mobile Device Protocol Stacks

Vijay T. Raisinghani

PhD candidate

Advisor: Prof. Sridhar Iyer

Committee: Profs. Abhay Karandikar, Anirudha Sahoo, Krishna Paul

K R School of Information Technology

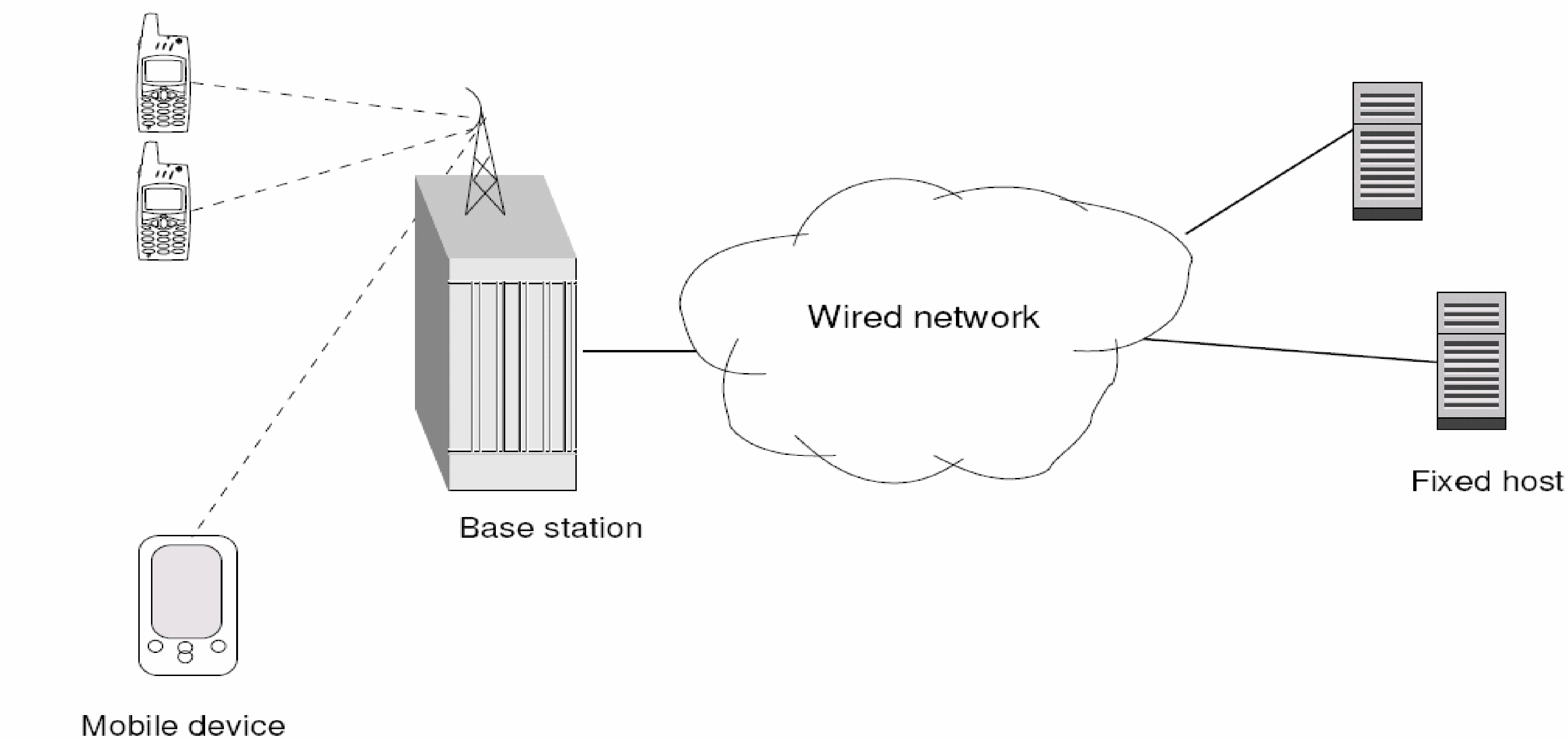
IIT Bombay

11-Dec-2006

Overview

- Background: Cross Layer Feedback
- Problem Definition: CLF architecture/related work
- ECLAIR Architecture
- ECLAIR Prototype/Validation
- ECLAIR Evaluation/Comparison
- ECLAIR sub-architecture/optimization
- Architecture Selection
- Future Work
- Publications

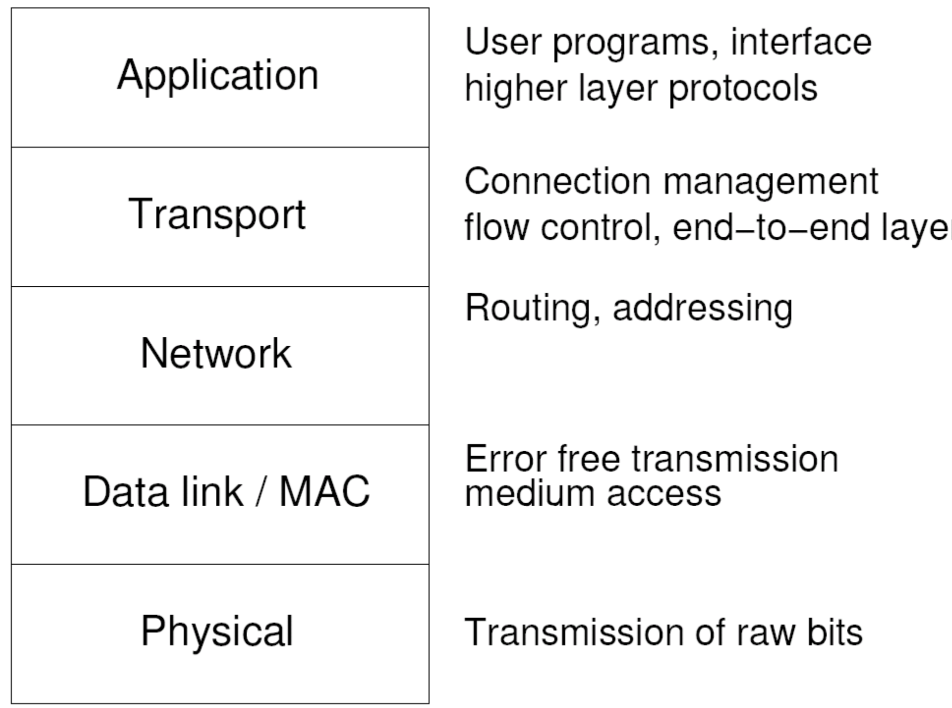
Typical Mobile Wireless Network



- MWN characteristics
  - High bit error rate of wireless channel
  - Mobility induced disconnections

Typical Protocol Stack Architecture - Layered

- Application has low awareness of physical layer and vice-versa
- Layered architecture: Layer *n* has function specific Service Access Points for layers *n* − 1, *n* + 1



Cross Layer Feedback: Motivation

- Protocol stack layering useful from software engineering perspective
- Strictly layered stacks do not perform well over wireless networks
  - network conditions are highly variable: random errors intermittent disconnection
  - Several assumptions from fixed wired networks do not hold for wireless, since packet losses, disconnections, mobility

Layered inefficiency example
TCP in Wireless

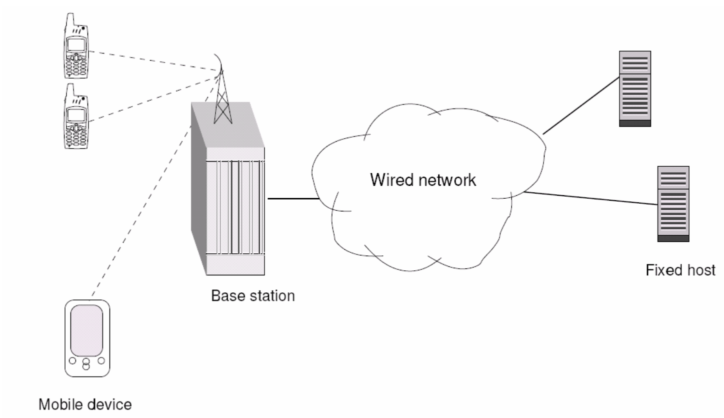
- On packet loss
  - TCP assumes network congestion
  - reduces throughput
- In wireless networks
  - many packet losses are due to bit errors
- TCP's congestion assumption fails
  - unaware of wireless physical layer
  - reduction in send window inappropriate

Cross Layer Feedback

- Cross layer information can help improve performance over wireless networks
- Upper to lower layers
  - TCP timer information
  - application QoS requirements
  - user feedback
- Lower to upper layers
  - link characteristics
  - network connectivity status
- Our study (Receiver Window Control) confirms the benefits of cross layer feedback

Cross Layer Feedback Optimizing for MWN

- Any cross layer approach involves one or more of:
  - Fixed Host (FH) TCP stack modification
  - Base Station (BS) per-connection support
  - Mobile Host (MH) TCP stack modification
- Our focus
  - Cross layer feedback on the MH



Overview

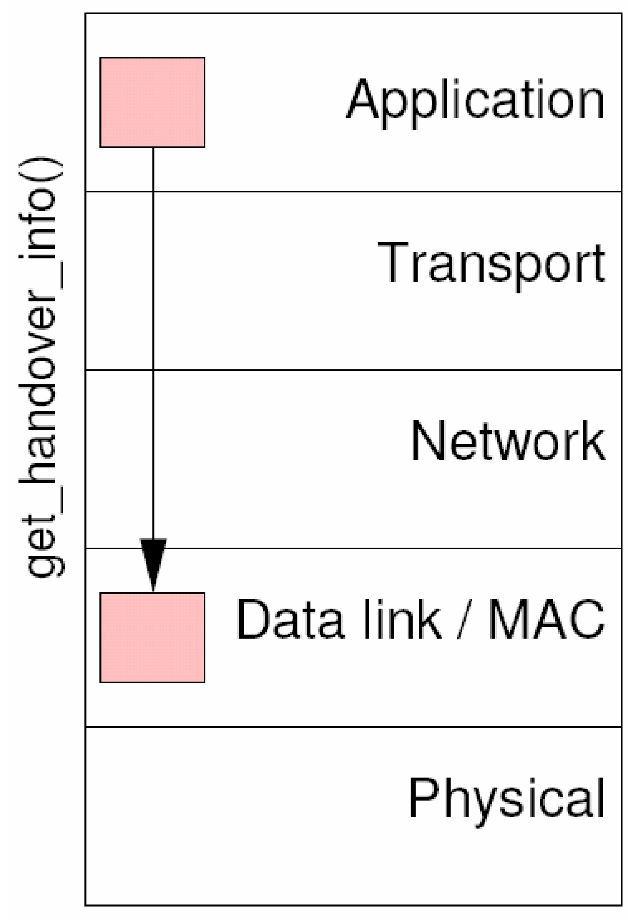
- Background: Cross Layer Feedback
- Problem Definition: CLF architecture/related work
- ECLAIR Architecture
- ECLAIR Prototype/Validation
- ECLAIR Evaluation/Comparison
- ECLAIR sub-architecture/optimization
- Architecture Selection
- Future Work
- Publications

Scope of Work

- Scope: How to do cross layer feedback; architectural aspects
- Out of scope
  - Specific cross layer optimization
    - Large body of literature exists on cross layer optimizations
  - Issues specific to cross layer feedback
    - Dependency cycles and conflicts

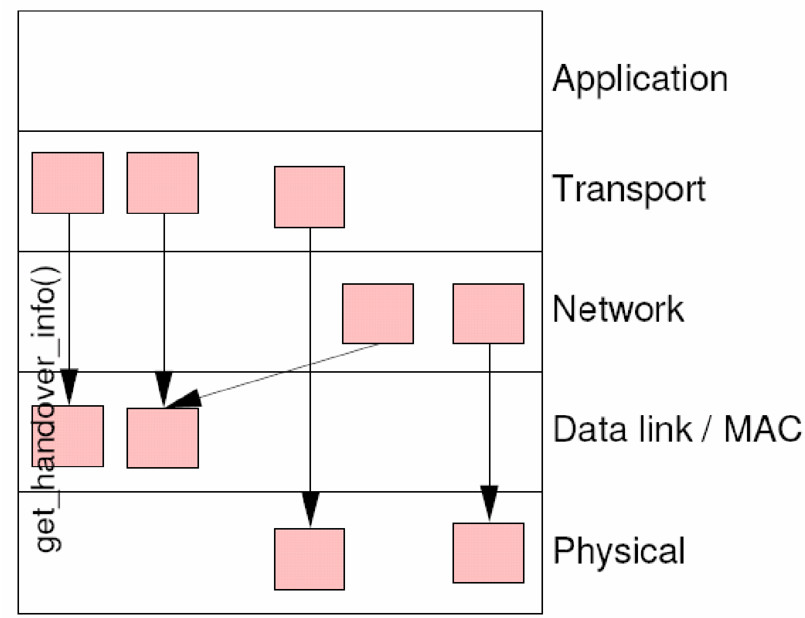
Cross Layer Feedback: "Punch hole" / Ad hoc approach

- Ad-hoc approach
  - Introduce additional code in layer for CLF



CLF: Punch Hole – Problems

- Each additional CLF code block can slow down data path (thruput) of layer
- Porting CLF will require rewriting for specific OS
- Difficult to disable/ remove code intertwined with regular layer code
- Difficult to do fast prototyping/additions since ad-hoc
- Multiple event monitors within a layer could slow down data path (thruput) of layer
- Difficult to control protocol's correctness since updates by different CLF code blocks



Existing Approaches

- Physical Media Independence (Inouye et al, 1997)
  - Adaptation modules for each layer
  - Layer by layer propagation of events
  - Operating System APIs for adaptation
- Interlayer Signaling Pipe (Gang et al, 1999)
  - Information exchange through packet headers; layers need to be modified
- ICMP Messages (Sudame et al, 2001)
  - Special ICMP messages and special handler at socket layer
  - Adaptation for application and transport defined by each application separately; layers need to be modified
- CLASS (Wang et al, 2003)
  - Direct interaction between layers; problems similar to ad hoc approach
- MobileMan (Conti et al, 2004)
  - Add network status data structure; rewrite protocols to be network aware
- User-space (Mehra et al, 2003)
  - All modules in user-space

Problem: CLF architecture

- CLF basically stack modification
  - Multiple ad-hoc cross layer modifications can impact stack's *efficiency, maintainability, correctness*
  - Existing approaches do not address all of these issues
  - Any to any layer interaction is not supported in all approches
- Problem: There is a need for an appropriate architecture for cross layer feedback
  - Design goals for architecture
    - Rapid prototyping: easy development / deployment of new CLF idea
    - Minimum intrusion: protect stack correctness; easy to extend / reverse CLF
    - Portability: easy porting to different systems
    - Efficiency: minimal overheads (e.g. cpu, memory, data path delay); enhanced performance
    - Any-Any layer communication: Any layer can communicate with any other layer in the stack

Contributions

- ECLAIR: Architecture for CLF
  - Definition, prototype implementation, validation(RWC)
- Core: Sub-architecture for reducing overheads
- Metrics for CLF architecture evaluation
- Notation for layer and CLF implementation aspects
- Design guide for cross layer feedback

Overview

- Background: Cross Layer Feedback
- Problem Definition: CLF architecture/related work
- ECLAIR Architecture
- ECLAIR Prototype/Validation
- ECLAIR Evaluation/Comparison
- ECLAIR sub-architecture/optimization
- Architecture Selection
- Future Work
- Publications

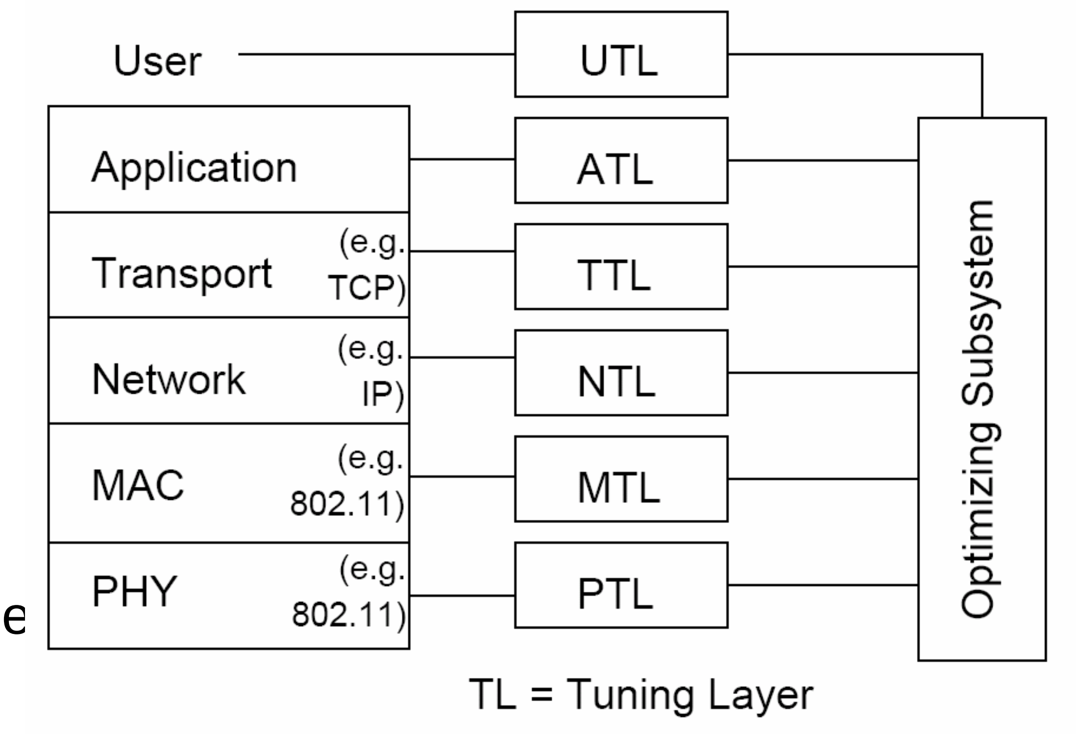
ECLAIR Motivation

Based on the design goals

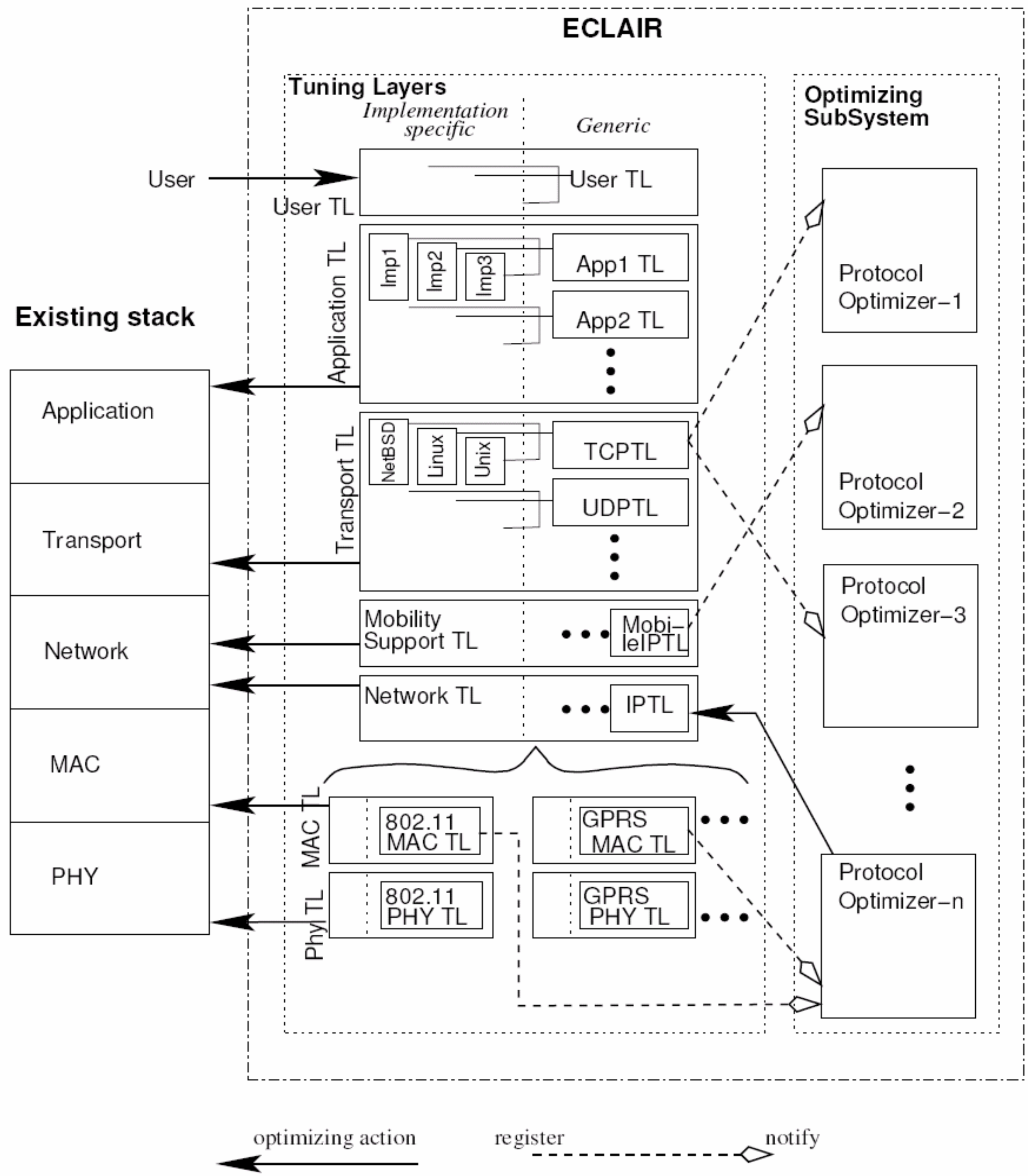
- Rapid prototyping
  - Provide clean hooks to enable quick changes in cross layer algorithms, without disturbing existing stack
- Portability
  - Provide APIs to reduce dependency on OS specifics
- Minimum intrusion
  - Use a mechanism to change protocol behavior with minimum possible modifications to existing stack
- Efficiency
  - Cross layer components should not impact the data path
- Any-to-any layer feedback
  - Components should not restrict direction of cross layer feedback or be restricted to specific layers

ECLAIR overview

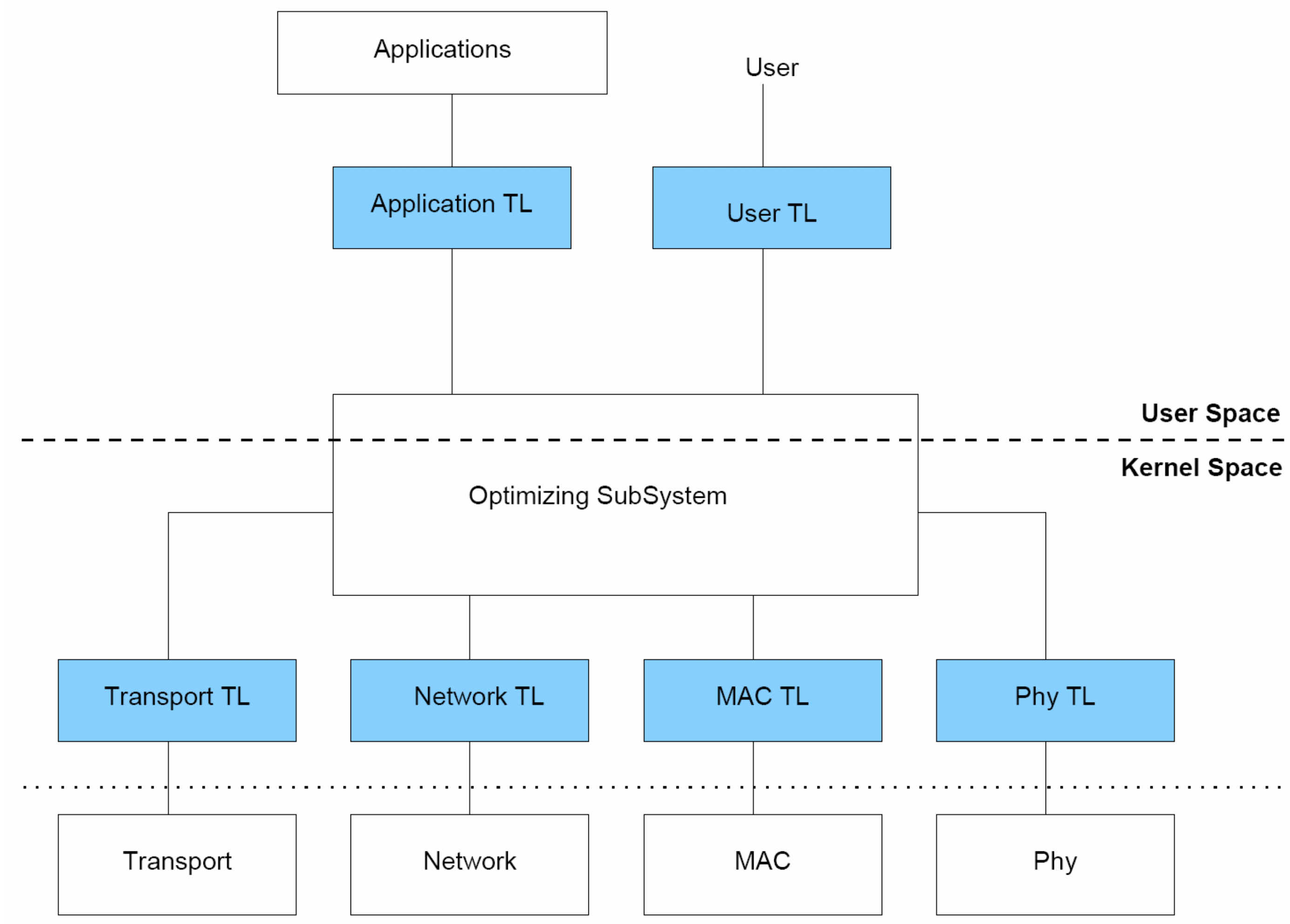
- Optimizing SubSystem: Protocol Optimizers (Cross layer feedback algorithms); receive layer events; decide other layers behavior
- Tuning Layer: Monitor layer events; provide API to protocol optimizer; access layer's control data structure values to monitor and change behavior



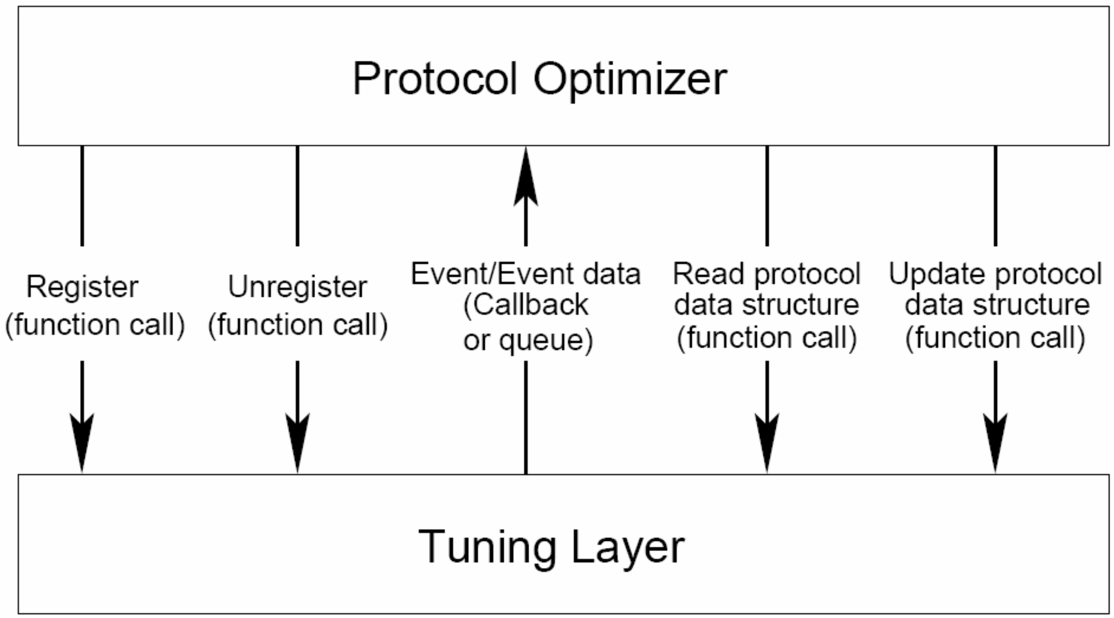
ECLAIR details

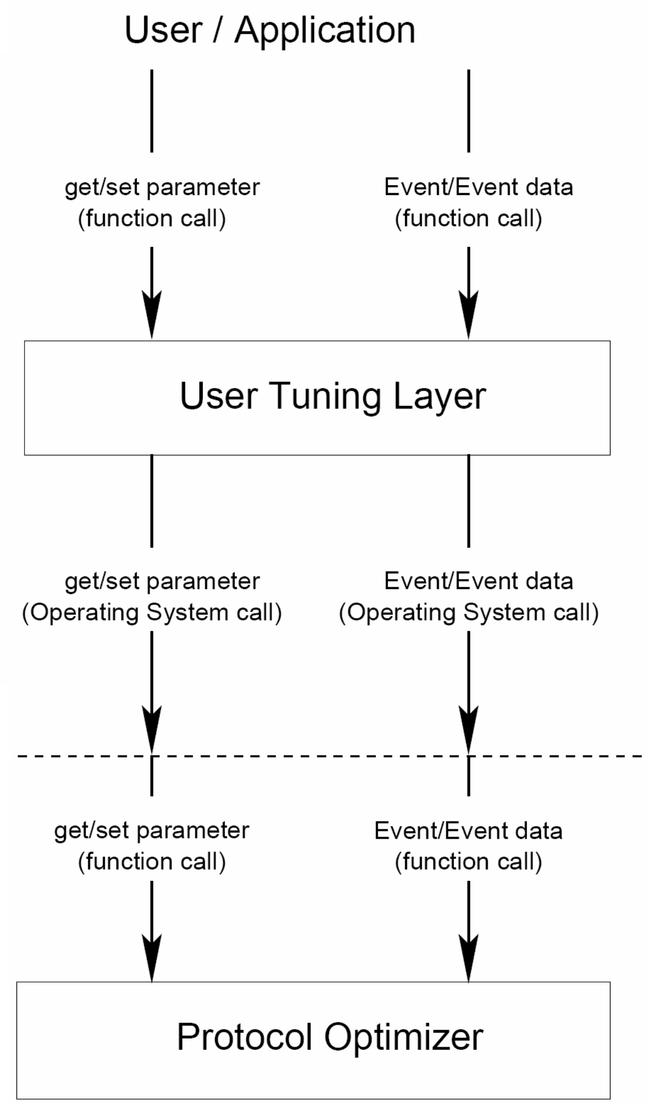


ECLAIR – Implementation View



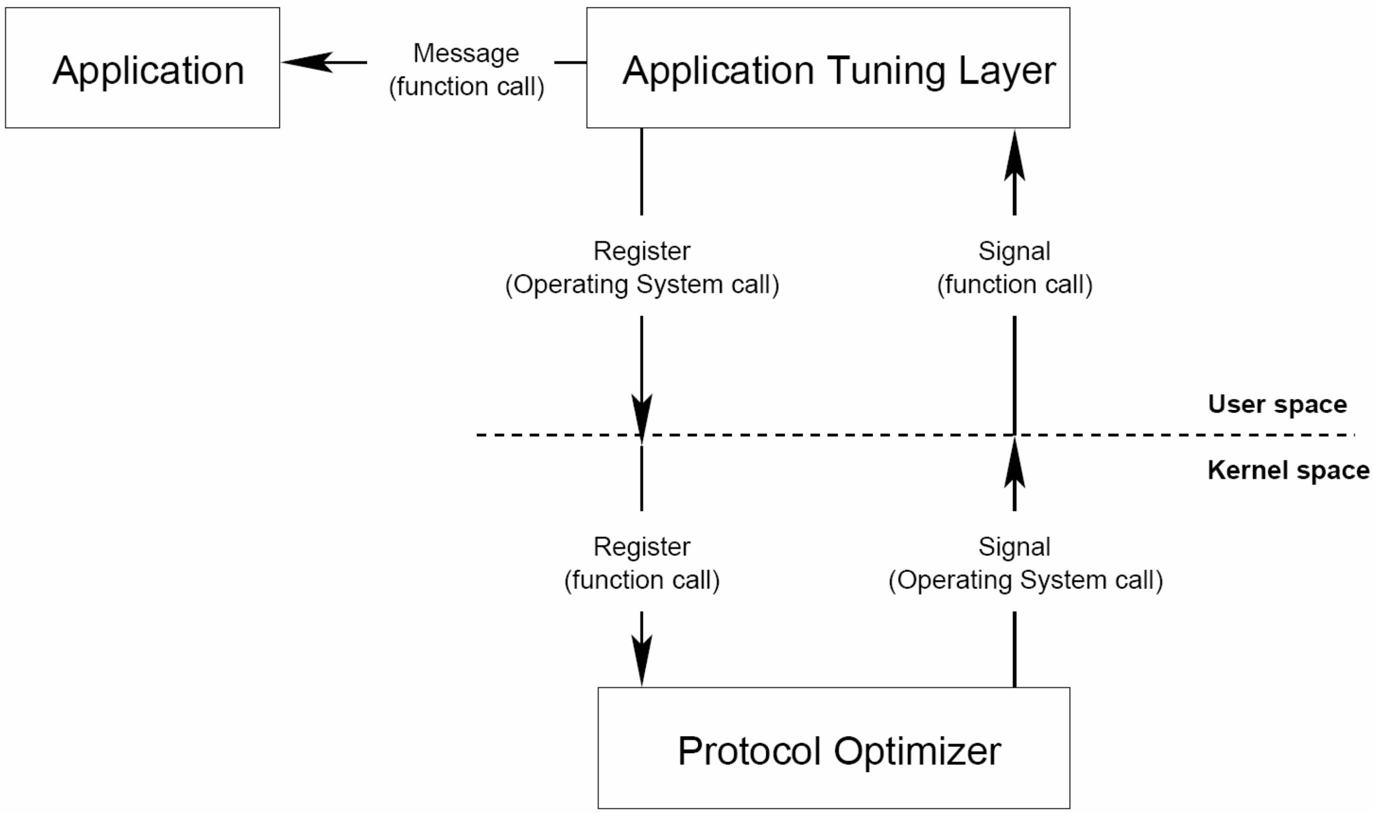
TL PO Interface





Protocol Optimizer for User TL

Application TL Interfaces



ECLAIR Details – Tuning Layer

- Application and User Tuning Layers are in *user-space*
  - Other TLs are in *kernel space*
- User TL (in user-space) interacts with system thru a special user-PO in kernel space (RWC example later)
- For portability
  - TL is split into *generic* tuning layer and *implementation dependent access sub-layer*
  - Generic API is used by PO; this invokes implementation specific API
- TL provides *register* and *unregister* functions for POs
  - Multiple POs may register for an event
  - *Event* means a change in some data structure of a protocol
  - Event notification to PO is through a call-back function or event queue
- If there is any *fatal* error during processing of a TL, the TL
  - Unloads (till next reboot)
  - Unregisters all its POs; aborts all its actions

ECLAIR Details – Optimizing SubSystem

- PO implements
  - Cross layer procedure (cross layer algorithm)
  - Event handler(s)
  - Register/unregister procedure, error handler, log procedure
- On receipt of event
  - Cross layer algorithm in PO determines values for updating data structures
  - PO calls appropriate TL APIs
- On *fatal* error
  - Unloads (unregisters from all TLs) until reboot
- Cross layer shutdown
  - User-PO registers with all TLs
  - On *shutdown* event from user, user-PO sends shutdown event to all TLs; thru TLs to all POs

ECLAIR TL API Examples

- User TL
  - get_stack_paramter() / set_stack_parameter()
  - crosslayer_shutdown()
- Transport TL
  - get_receiver_window() / set_receiver_window()
- Network TL (IP)
  - get_frag_assy_timer() / set_frag_assy_timer()
- MAC TL (802.11)
  - get_contention_window() / set_contention_window()

ECLAIR Validation
Receiver Window Control: CLF example

- TCP congestion control: sender – *congestion window*; receiver – *receiver / advertised window*
- Manipulate *receiver* window to manipulate throughput of flows
- At receiver
  - Flows with similar *rtt* and bandwidth on path get similar throughput, assuming no congestion
  - Reduce advertised window of low priority flow to decrease throughput

Receiver Window Control – Algorithm

Events

UserFeedback: User changes priority of applications

Initialization

n = `number of applications`
B = `available bandwidth` // constant
R = `rtt` // constant
$A = B \times R$
$awnd_i$ = `default advertised window` // initially all equal
$x_i = 1$ // application priority; initially all equal

Function Calls

ReceiverWindowControl()

$cum\_awnd = 0;$
// User changes priority of applications; x_i is changed
for $i = 0$ to $i < n - 1$ **do**
  $awnd_i = round(\frac{x_i}{\sum_{j=0}^{n-1} x_j} * A)$
  $cum\_awnd = cum\_awnd + awnd_i$
end for
$awnd_{n-1} = A - cum\_awnd$

RWC Example

- Initially

  $A = 30, x_1 = 1, x_2 = 1, x_3 = 1$

  - Thus

  $awnd_1 = 10, awnd_2 = 10, awnd_3 = 10$

  - After user feedback

  $x_1 = 2, x_2 = 1, x_3 = 1$

  $$awnd_1 = round(\frac{2}{4} \times 30) = 15$$

  $$awnd_2 = round(\frac{1}{4} \times 30) = 8$$

  $$awnd_3 = 30 - (15 + 8) = 7$$

ECLAIR Prototype: Linux Receiver Window Control

1: Get TCP control data structures

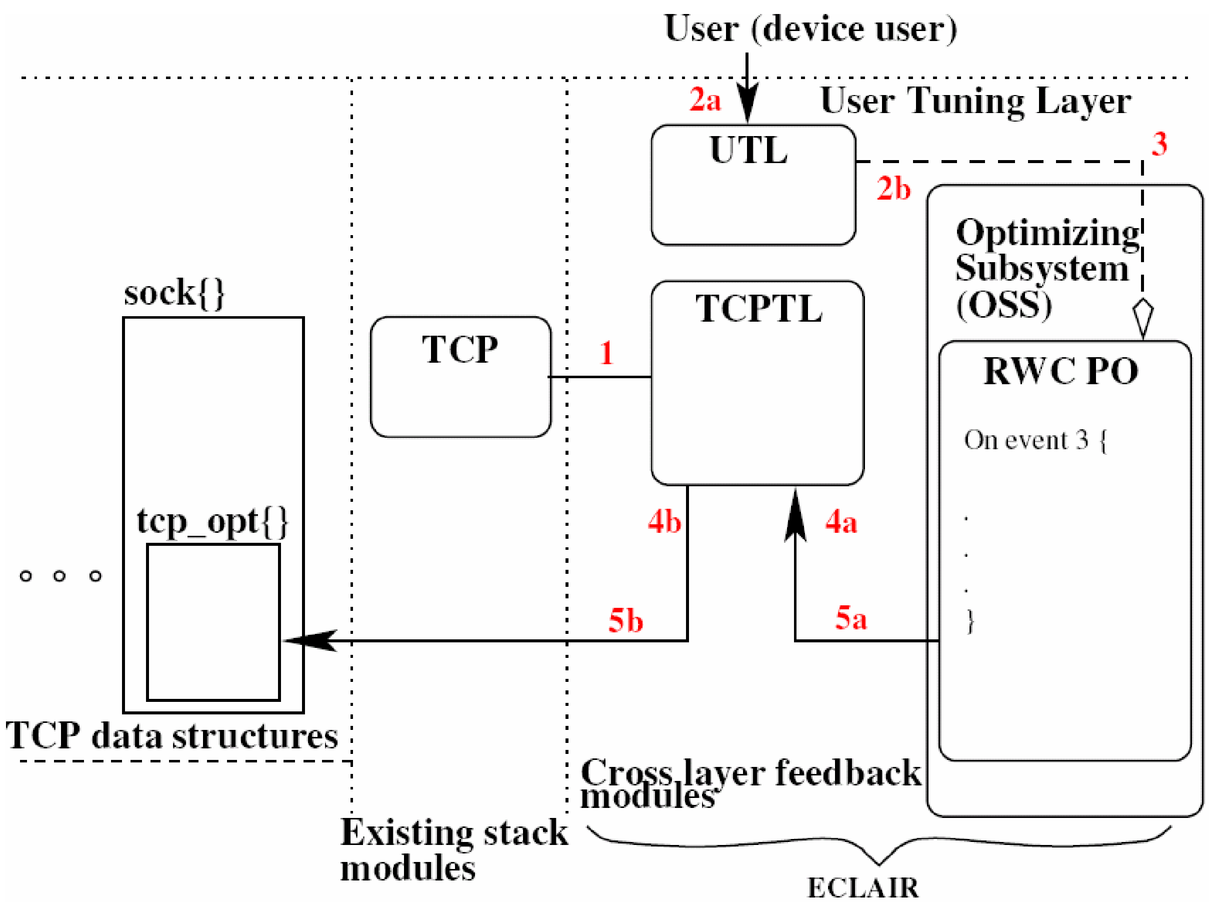
2a: PO registers

2b: User event

3: App info, user inputs to PO

4a,b: PO reads receiver window values via TCPTL

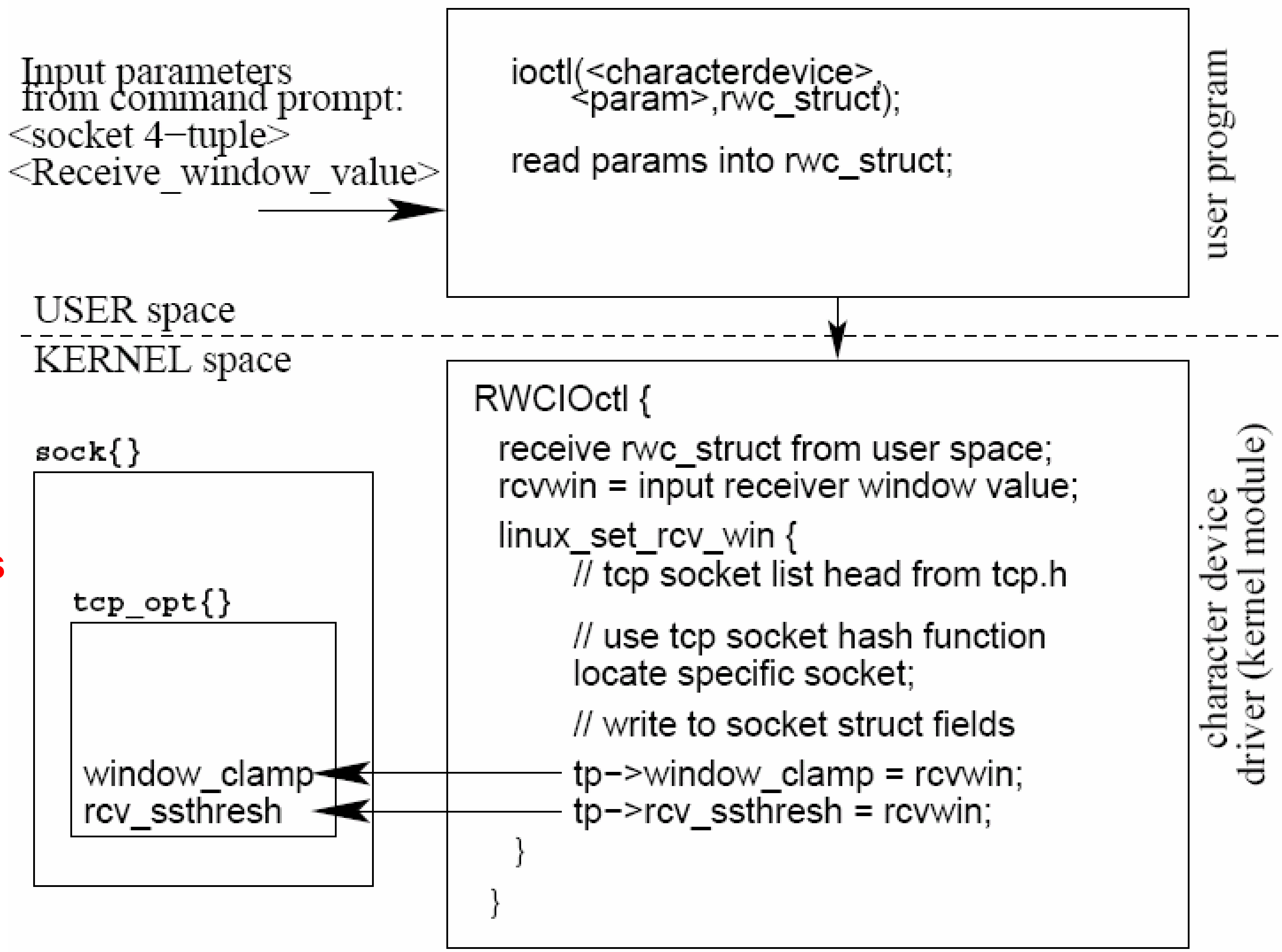
5a,b: PO sets receiver window values via TCPTL



ECLAIR Prototype: Linux (contd..) Receiver Window Control

- Source code search tools used
  - CScope
    - Source code indexing and search utility
  - CBrowser
    - Front end for cscope
  - Linux Cross Reference web-site
    - Source code indexer and viewer via web-browser
- Identified relevant TCP code and data structures
  - Receiver window control manipulation points
  - Relevant data structures/variables window_clamp, rcv_ssthresh

ECLAIR Prototype: Linux (contd..)
Receiver Window Control



No changes to existing stack

RWC Code

```
1    struct tcp_opt {
       .
       .
                    /* Maximal window to advertise */
2                   __u32   window_clamp;
                    /* Current window clamp        */
3                   __u32   rcv_ssthresh;
       .
     }

4    struct sock {
       .
       .
5                union {
6                          struct tcp_opt          af_tcp;
                           .
                           .
                       }
     }
```

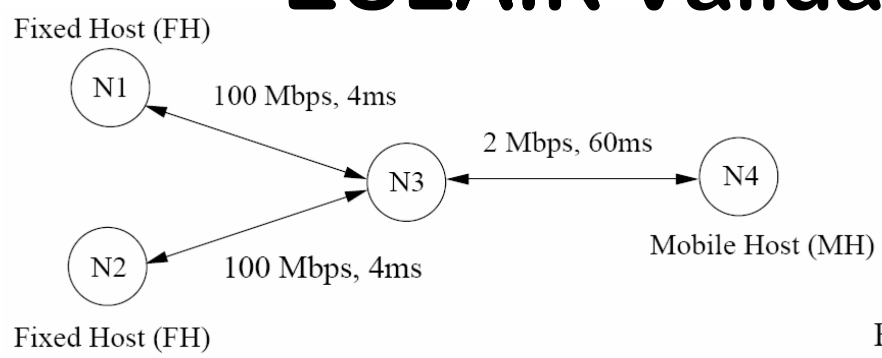
RWC Code

```
#define IOCTL_MAJOR 250
#define IOCTL_GETVALUE 0x0001
...
static int RWCIOctl(...)
{
    ...
    switch( cmd ) {
    case IOCTL_GETVALUE:
        rwc_ioctl_param = (struct rwc_info_struct *) arg;
        // copy socket 4-tuple to socket_info
        socket_info.s_addr =
                      rwc_ioctl_param->in_ip_addr.s_addr;
        ...
        rwc_window = rwc_ioctl_param->rwc_window;

        // TCP TL function
        linux_set_recv_win (socket_info, rwc_window);
    ...
}
static struct file_operations ioctlFops = {
...
    ioctl: RWCIOctl,  /* ioctl */
};
```

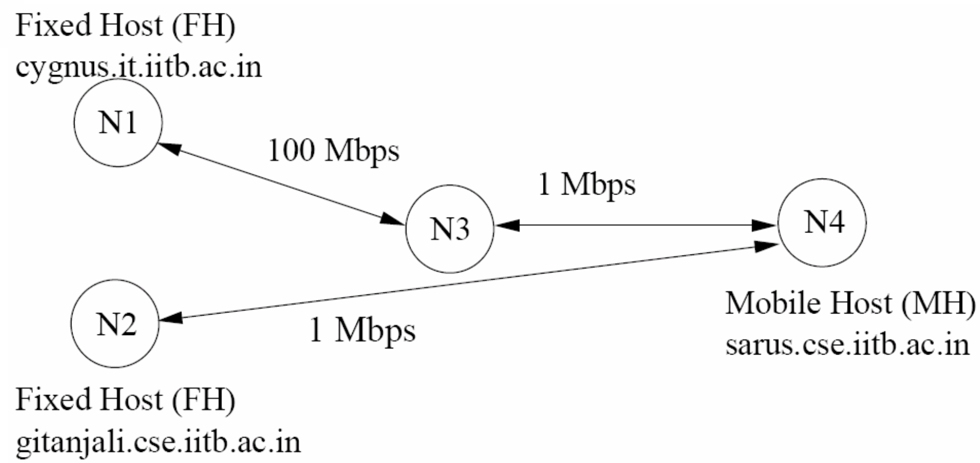
RWC Code

```
11    static int __init IoctlRWCInit(void)
      {
          printk(KERN_ERR "TCP RWC func load.\n");
12        if(register_chrdev(IOCTL_MAJOR,
                  "ioctl-rwcDriver", &ioctlFops) == 0) {
          ...
          };
          printk("ioctl: unable to get major %d\n",IOCTL_MAJOR);
          return( -EIO );
      }
----------------------------------------------------------------
13    linux_set_recv_win (socket_info, rwc_window)
      {

14        // locate socket using socket_info and copy of tcp_hashfn
15        struct tcp_opt *tp = &(sk->tp_pinfo.af_tcp);
16        lock_sock(sk);
17        tp->window_clamp = rwc_window;
18        tp->rcv_ssthresh = rwc_window;
19        release_sock(sk);
      }
```

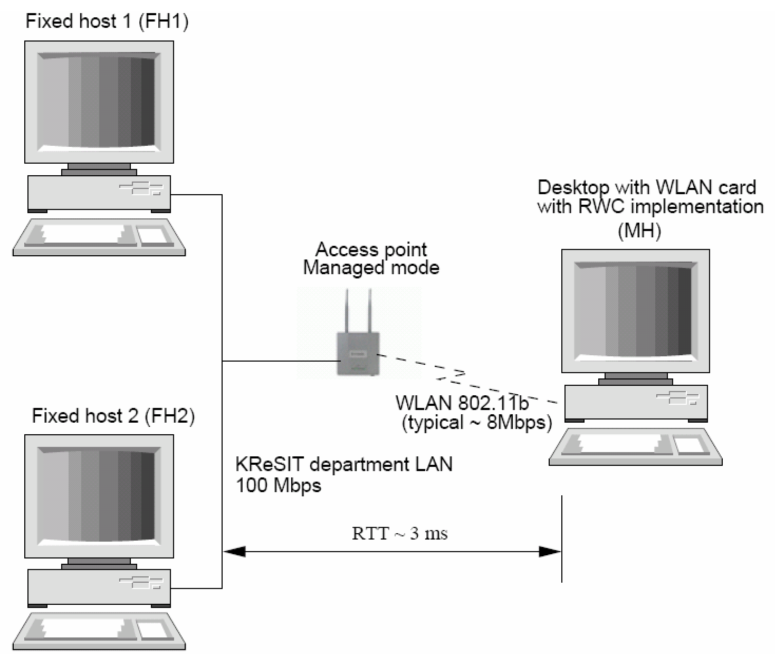
ECLAIR Validation using RWC



Step1: RWC Simulation: ns-2

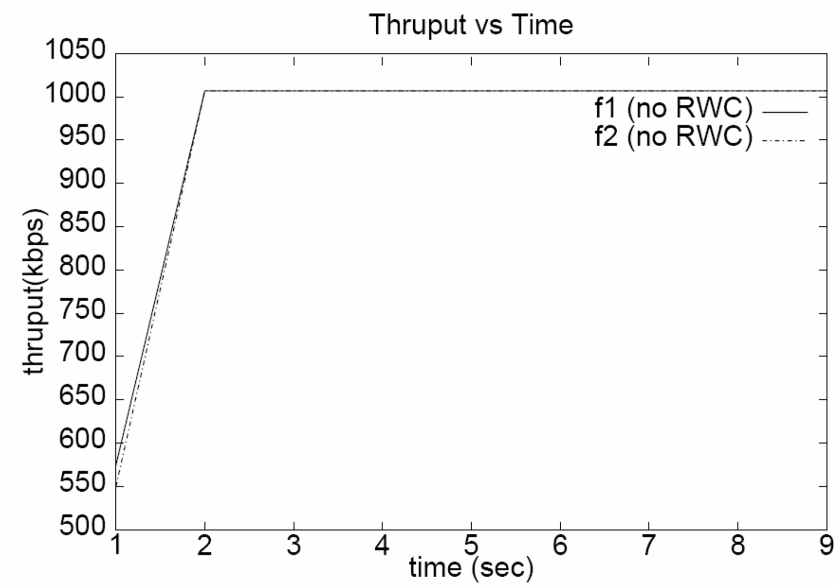


Step2: RWC Wireline Experiment

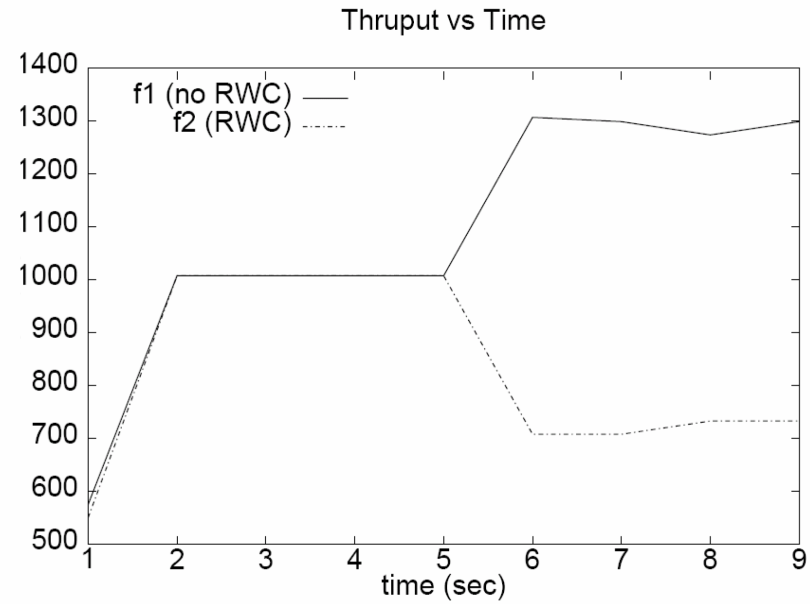


Step3: RWC Wireless Experiment (802.11)

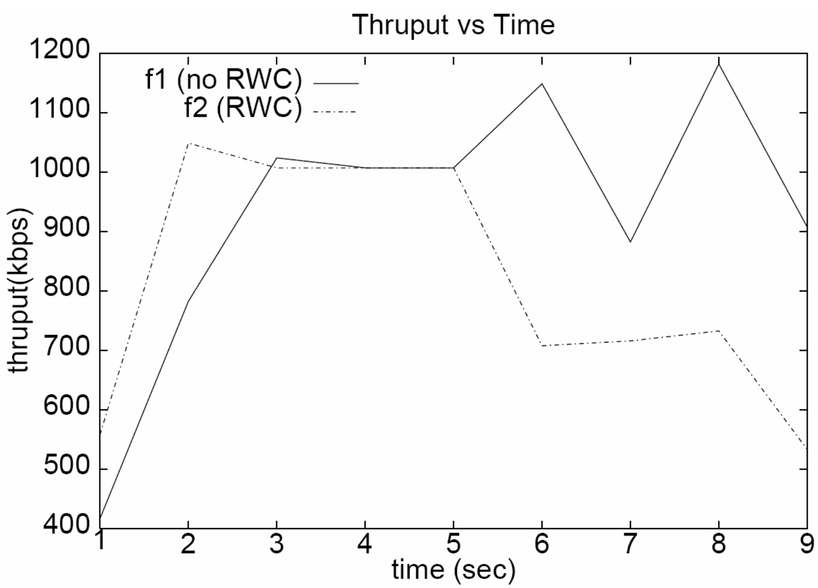
RWC Simulation Results



No RWC



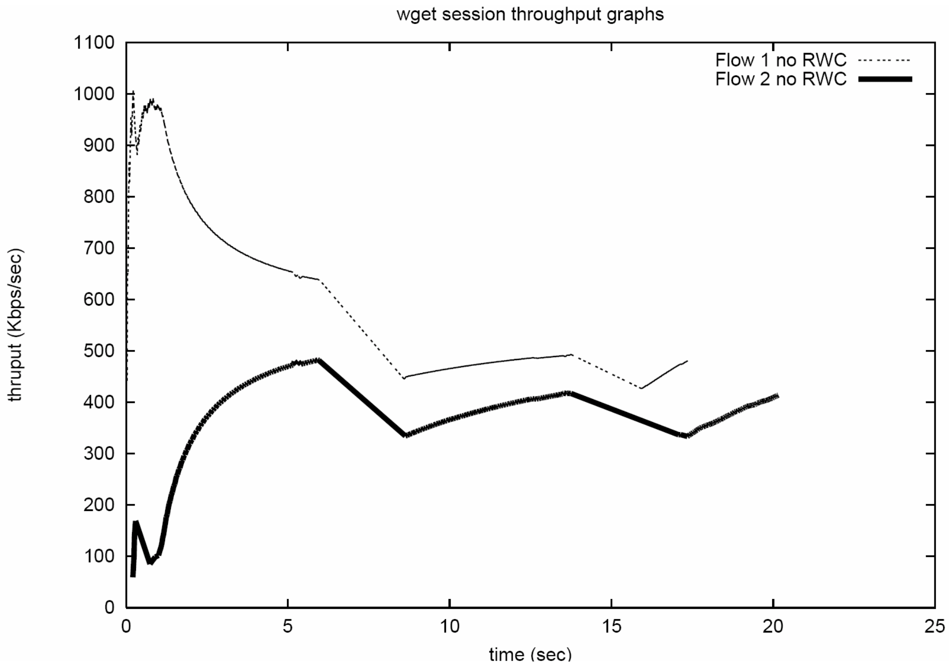
RWC
No pkt
loss



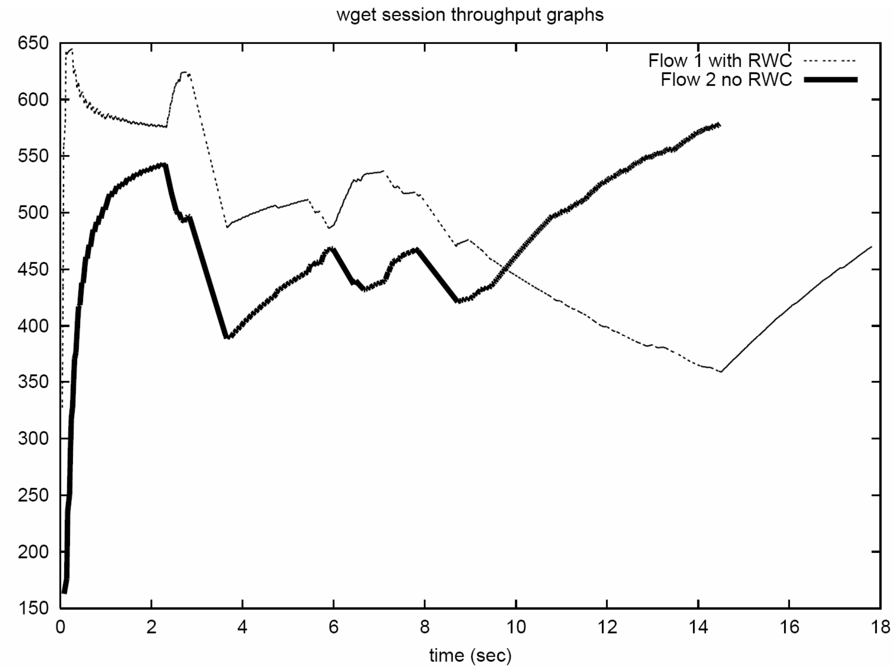
RWC

0.1%
pkt
loss

RWC Wireline Results

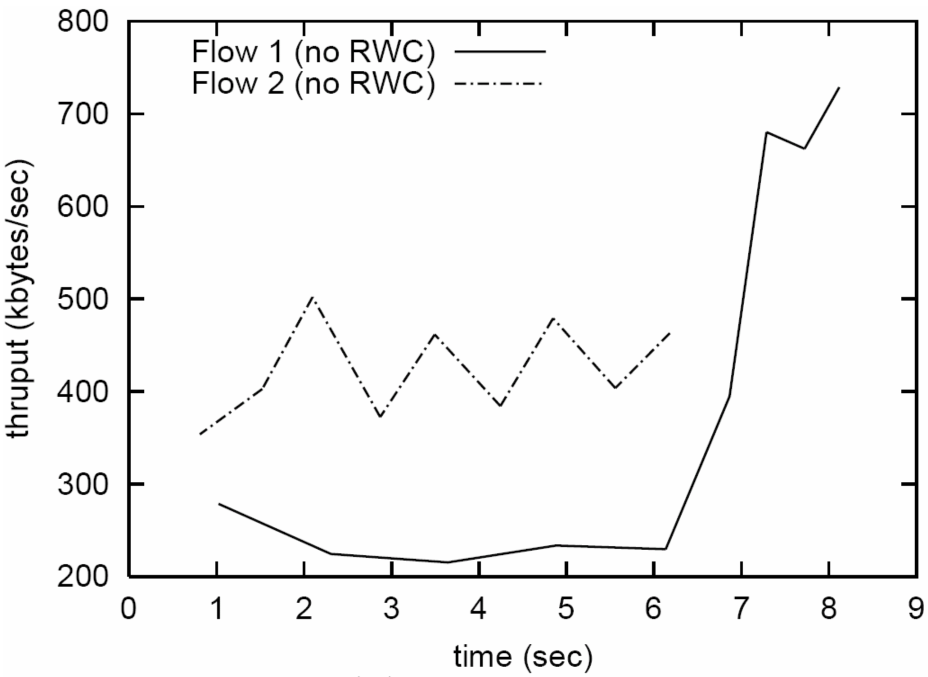


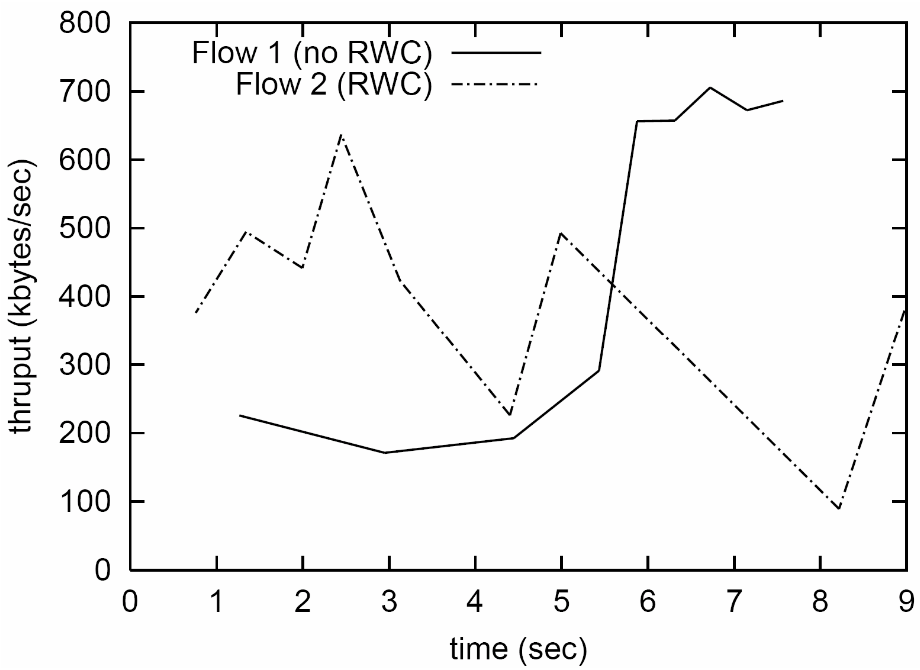
No RWC



RWC, invoke time = 8sec

RWC Wireless Experiment





No RWC

RWC; Window=2KB; at 5 sec

RWC Wireless Experiment- Mean/std dev table

		$Flow\ 1$	$Flow\ 1$	$Flow\ 2$	$Flow\ 2$
RWC invoked at (seconds after start)	Receiver Window Size (bytes)	$\bar{X}$	σ	$\bar{X}$	σ
		$kbytes/s$	$kbytes/s$	$kbytes/s$	$kbytes/s$
Not invoked	–	304·06	23·77	383·64	23·20
1	2048	543·42	15·76	243·36	3·96
2	2048	488·58	35·14	252·82	1·75
5	2048	347·93	4·98	277·98	6·03
1	1024	547·37	20·21	8·30	·37
2	1024	521·23	29·13	8·95	·64
5	1024	379·13	18·92	21·44	8·18
1	512	580·60	19·57	4·03	·034
2	512	541·45	26·40	4·44	·36
5	512	413·89	20·91	8·18	1·61

ECLAIR Validation Results

- Wireline and wireless experiment results using RWC modules
  - In-line with simulation results
- ECLAIR prototype works as expected
- Confirms ECLAIR prototype does not seem to introduce any new errors

Differences: Experiment v/s simulation

- Simulation *ftp* flows, stopped after 9 sec
- *wget* transfers in experiment
- WLAN no RWC
  - First flow gets most of the bandwidth, due to WLAN characteristics
- In experiment, throughput of controlled flow remains low
  - Receiver window value not reset
- Differences not significant
  - Do not impact validation results

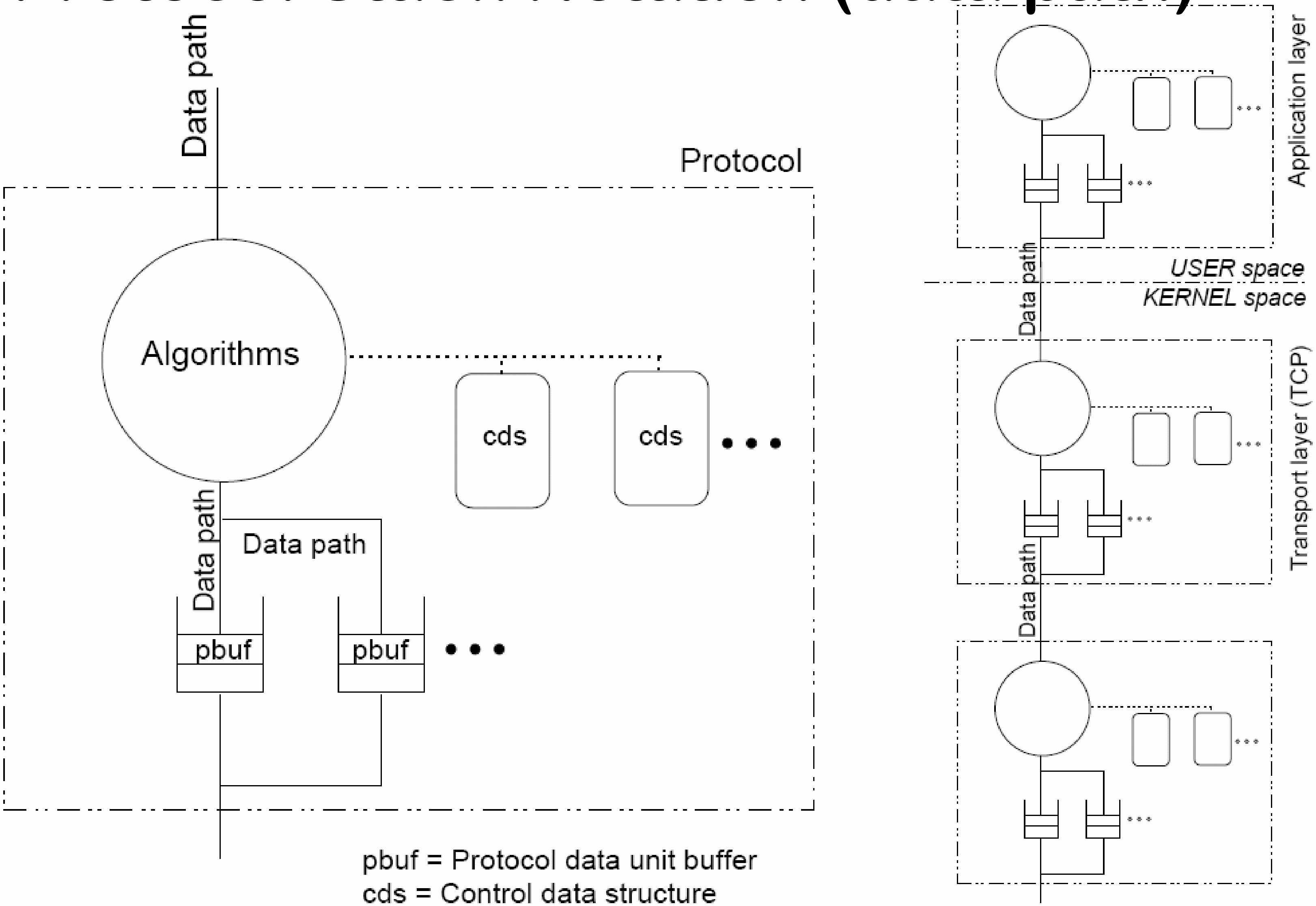
Overview

- Background: Cross Layer Feedback
- Problem Definition: CLF architecture/related work
- ECLAIR Architecture
- ECLAIR Prototype/Validation
- ECLAIR Evaluation/Comparison
- ECLAIR sub-architecture/optimization
- Architecture Selection
- Future Work
- Publications

Performance Metrics

- Design goals: rapid prototyping, minimum intrusion, portability, efficiency, any-any layer communication
- We propose following metrics for evaluating cross layer architectures
- Efficiency metrics
  - Time / space(runtime/footprint) overhead
  - User-kernel crossing
  - Data path delay
- Maintainability metrics
  - Rapid Prototyping: Effort required to add or modify cross layer optimization
  - Degree of intrusion: Impact points within the existing stack
  - Portability: Impact points within the cross layer optimization
- Any-to-Any layer interaction: Subjective assessment

Protocol Stack Notation (data path)

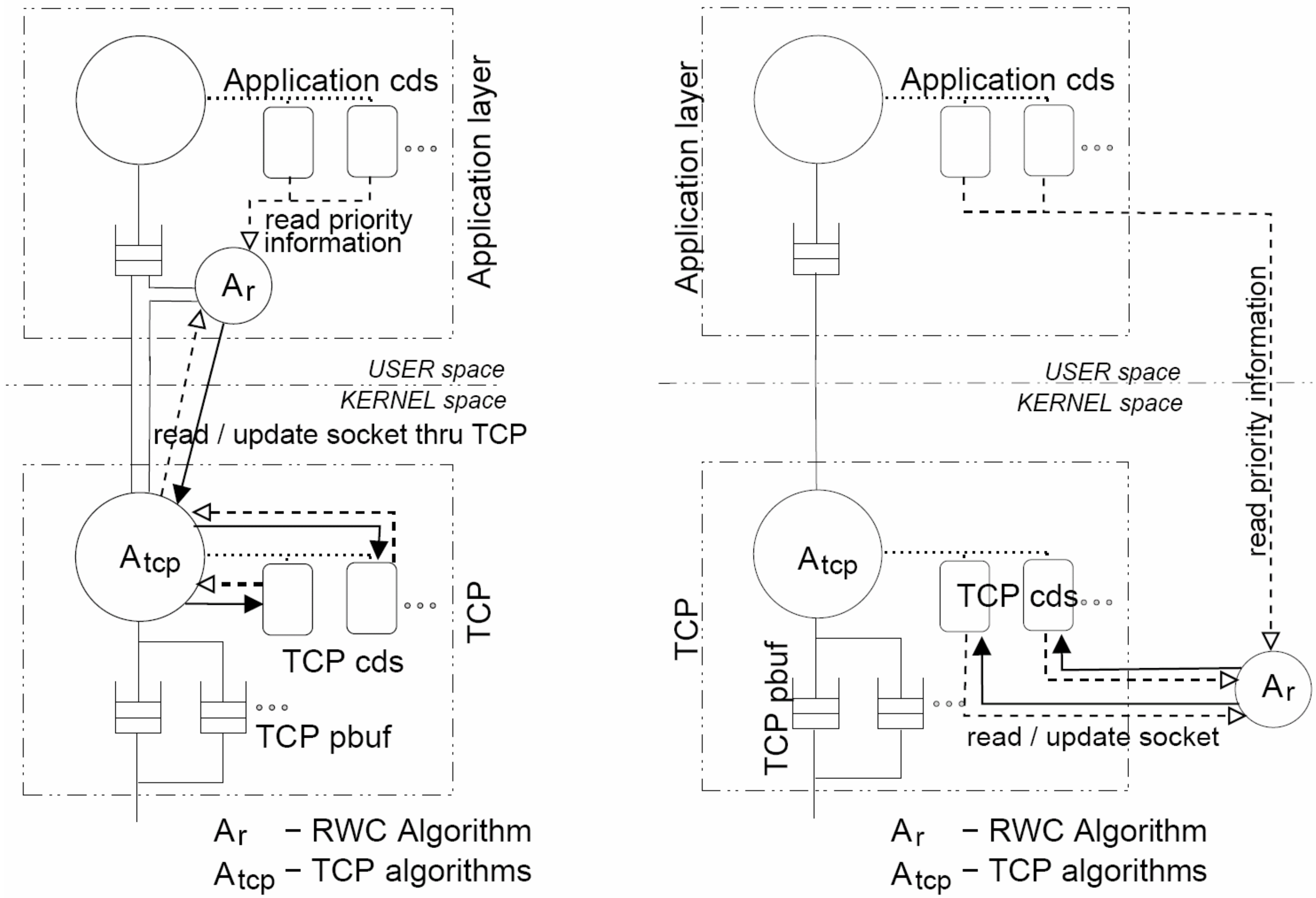


(a) Single protocol implementation (b) Protocol stack

Evaluation Approach

- Implementations of architectures not available
- Comparison through analysis
  - ECLAIR and user-space RWC
- Quantitative comparison through simulations
  - Data path delay
  - User-kernel crossing
  - Time/space overhead evaluation not possible without implementation
- Simulation of
  - Modification to protocol stack (archs like MobileMan, ISP, CLASS, ICMP, User-space on data-path)
  - User space implementation (user-kernel crossing impact)
  - ECLAIR
- Qualitative comparison
  - All metrics

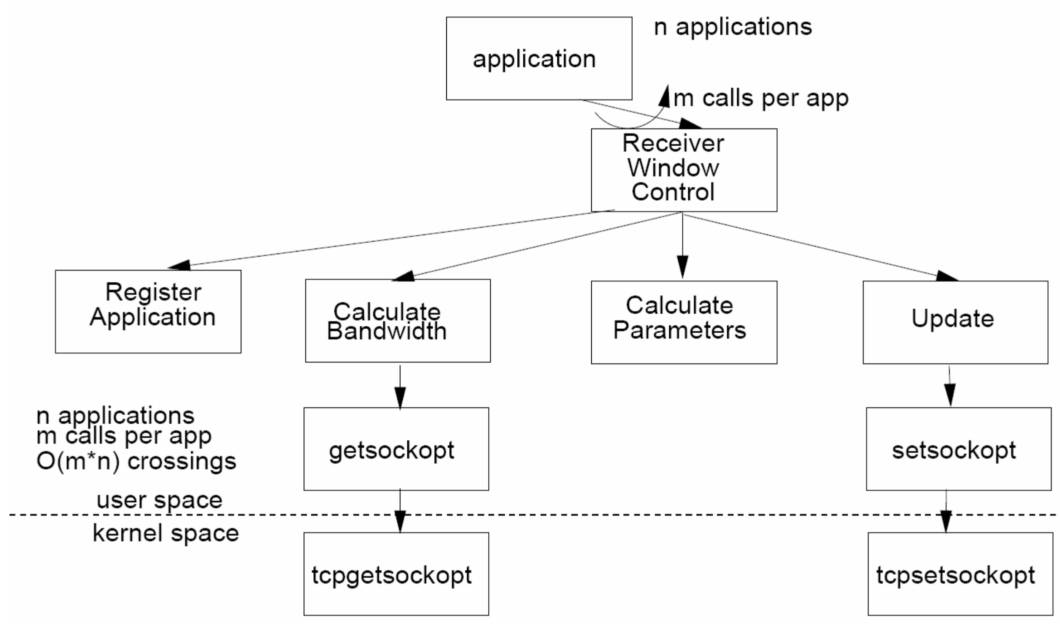
ECLAIR v/s User-space



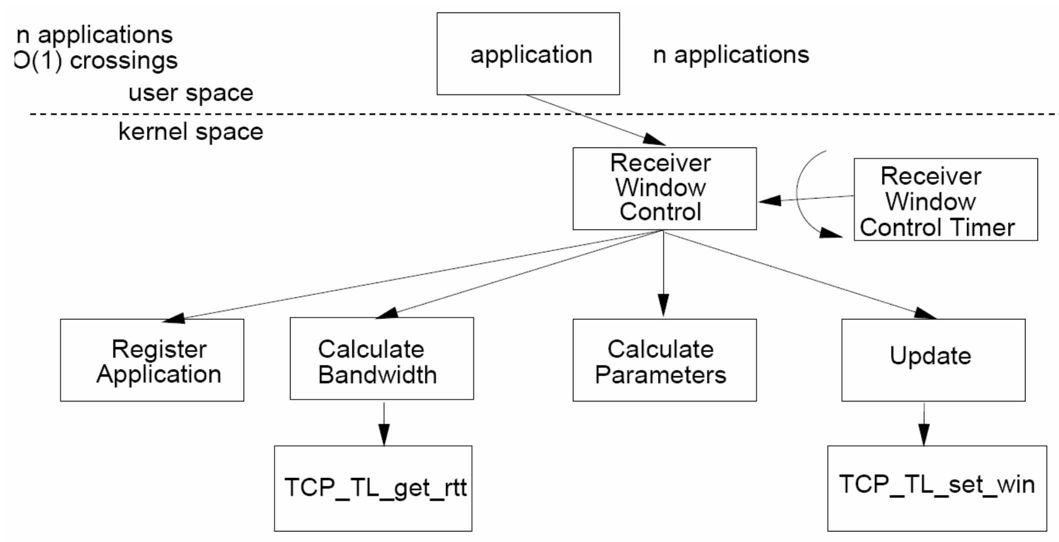
(a) User-space implementation (b) ECLAIR implementation

RWC Structure Charts

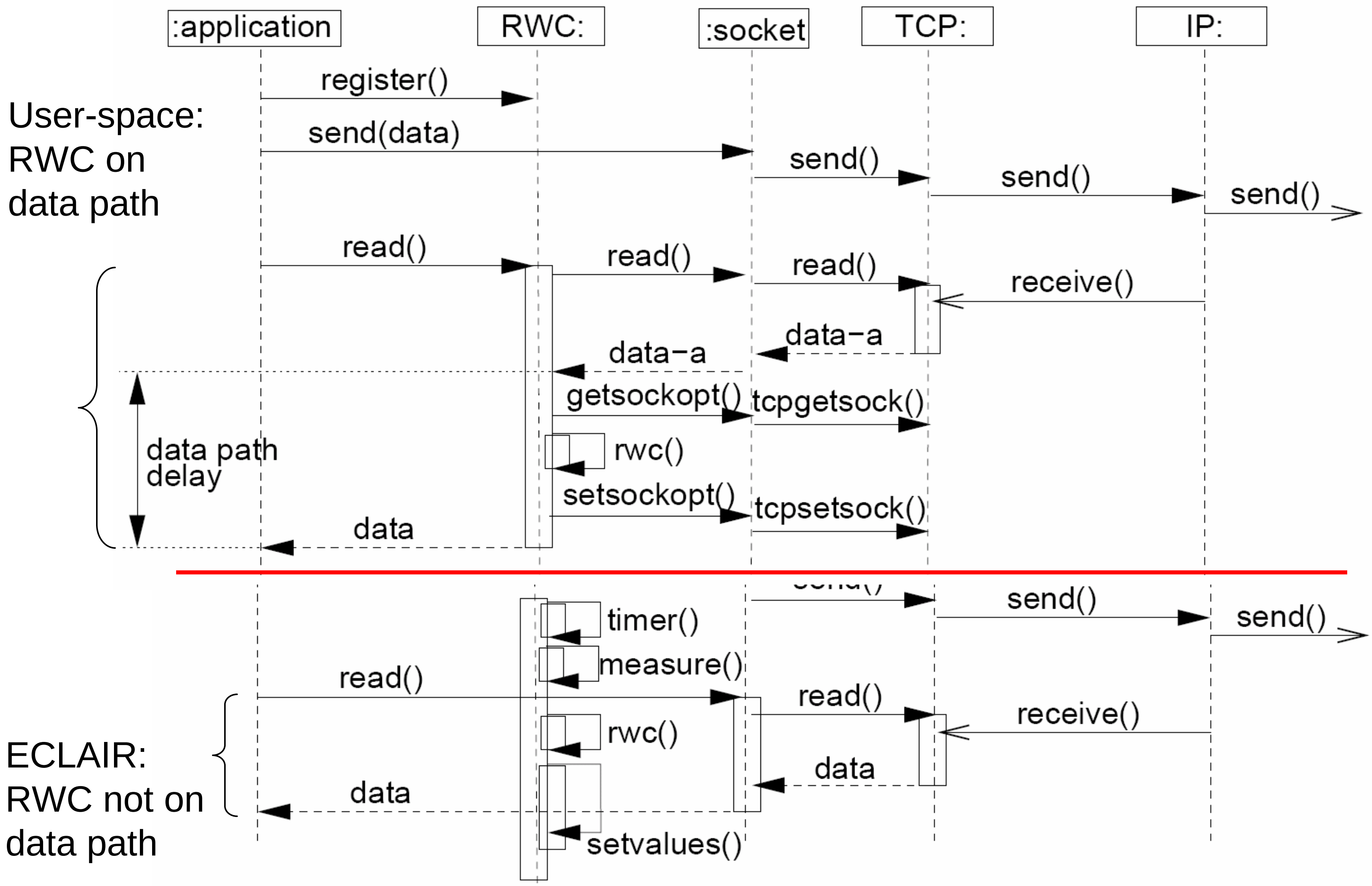
- User-space RWC



- ECLAIR RWC



RWC Sequence Diagrams

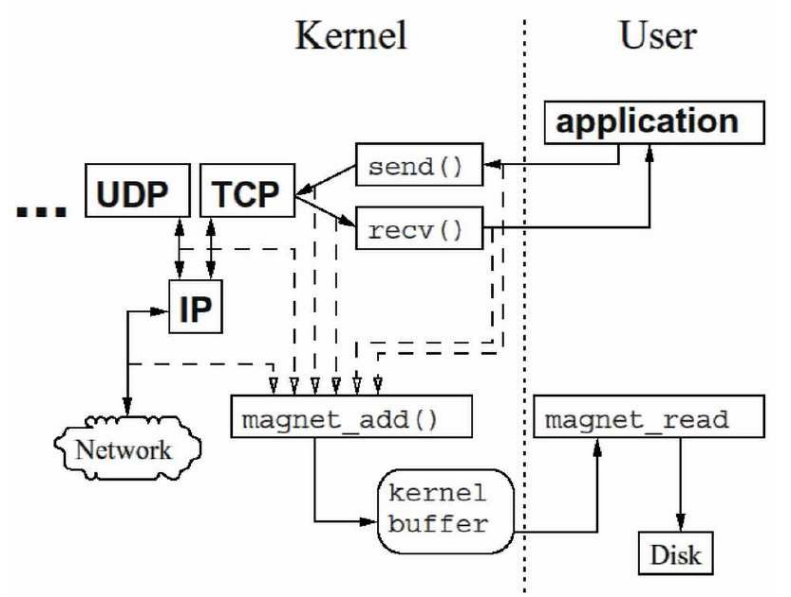


ECLAIR v/s User-space comparison

Evaluation metric	ECLAIR	User-space	Description
Time overhead (complexity)	O(t x n)	O(n x m)	n = number of applications t = no. invocations of RWC (ECLAIR) m = no. invocations of RWC (User-space), once for each *receive*
Space overhead	O(n)	O(n)	Space complexity: space for app information
User-kernel crossing	O(1)	O(n x m)	ECLAIR: using ioctl() User-space: getsockopt(), setsockopt()
Data path delay	--	O(n x m)	

Relative Overhead Measurement
Kernel Instrumentation

- Tools evaluated: MAGNET (Monitoring Apparatus for Generic kerNel Event Tracing), LTT (Linux Trace Toolkit), OProfile
- MAGNET
  - Traces packet movement within stack
  - *hooks* placed in link, IP, TCP
  - Uses CPU cycle counter
- LTT
  - Cannot trace packet movement
  - Allows creation of user defined events
- OProfile
  - Continuous overhead profiler
  - Regularly samples CPU registers
  - Statistical reports about programs executed
  - Cannot trace packet movement



Kernel Instrumentation

- Kernel 2.4.19 used; initial experiments done with
  - MAGNET and LTT (both patches applied)
    - Resulted in extremely large variations in packet movement time; discarded
  - Only OProfile patch
    - Not useful; discarded
  - Only MAGNET patch
    - Results were reasonably consistent
- Data path delay: MAGNET used for tracing packet movement within kernel
- User-kernel crossing: for sub-μsec measurements – get_cycles() used within application and kernel

Design of Experiments

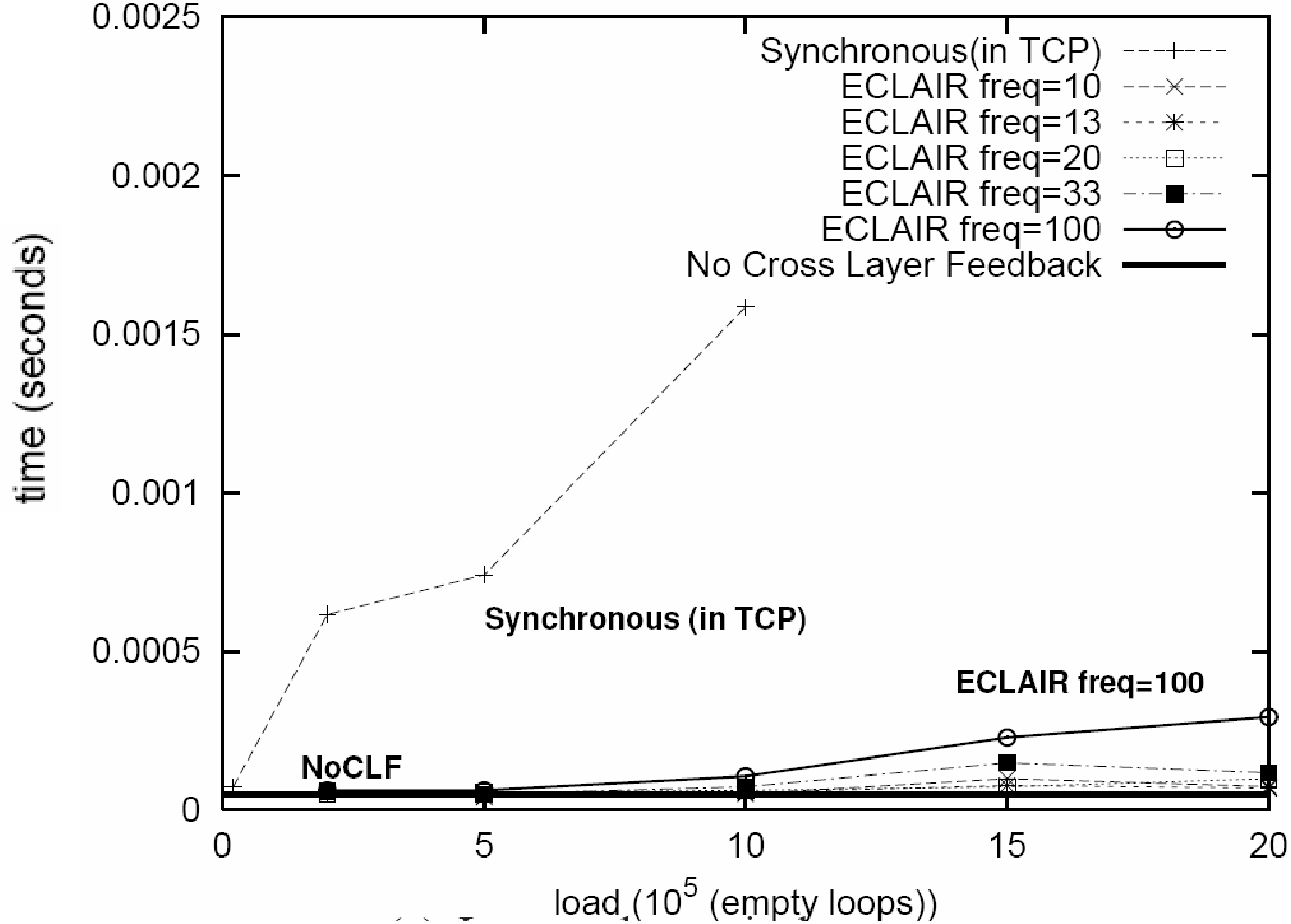
- Stack activity
  - Single download initiated from web, using wget
  - Two sites – low throughput, high throughput (increased stack activity)
- Cross layer overhead simulated by empty loop
  - 2, 5, 10, 15, 20 (x 10^5 cycles)
- ECLAIR overhead
  - Empty loops (dummy *for* loop) within RWC module
  - Kernel module loaded, loop executed within module – multiple times, module unloaded
  - Loop invoked at different frequencies, 10 to 100 times per second

Design of Experiments

- Protocol modification
  - Empty loops within TCP (load on data path)
  - tcp_v4_rcv() in TCP receive path modified
  - Loops executed within function i.e. for each packet
- User-kernel crossing with TCP socket search
  - User-space architecture
    - Operating system API used – setsockopt()
  - ECLAIR RWC
    - RWC module with tcp_hashfn() used to reduce search time
  - TCP socket hash collisions created to measure impact of change in bucket depth
  - CPU cache invalidated to rule out savings due to cache

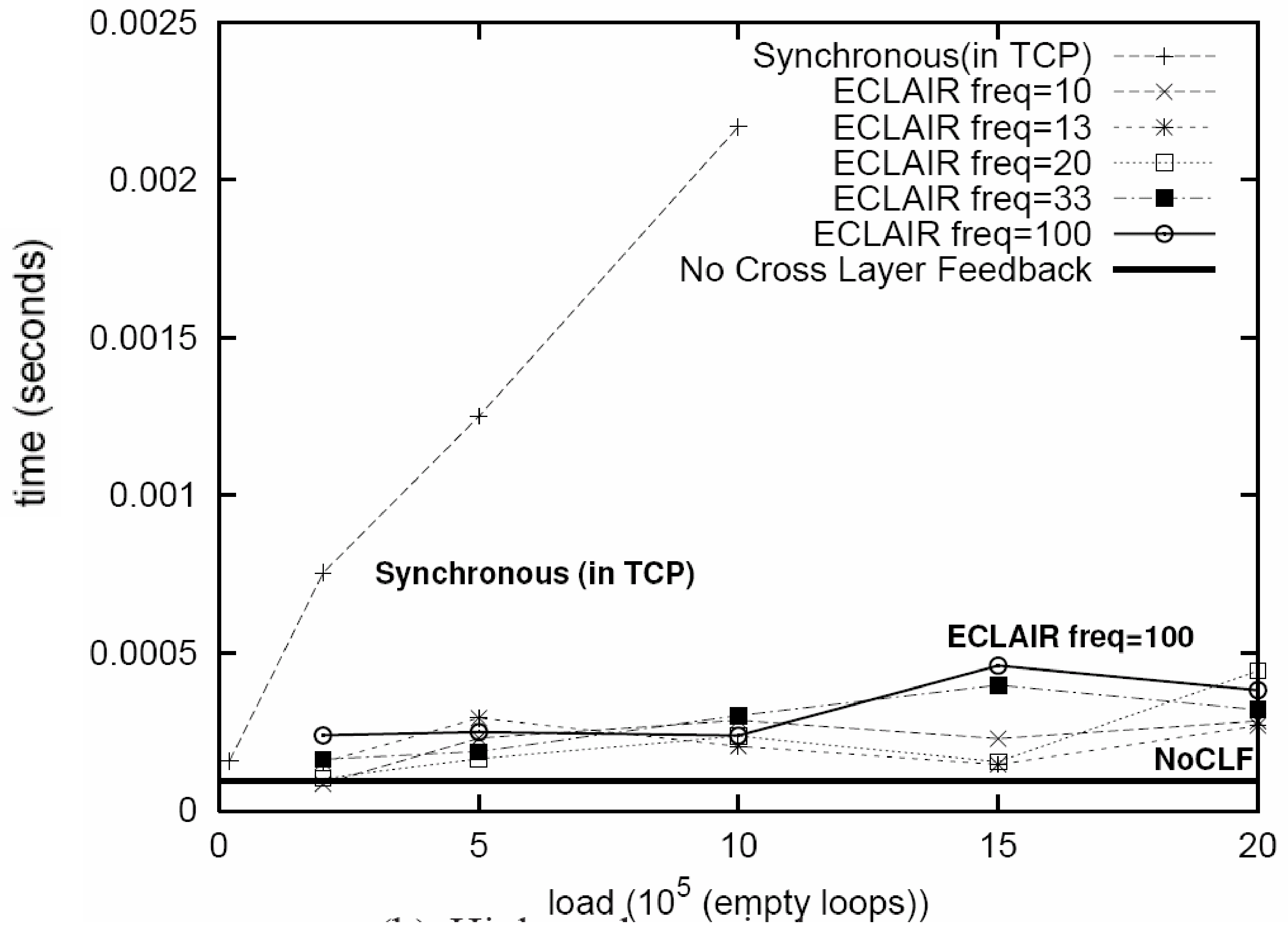
Data Path Delay

(Low packet rate)



Data Path Delay

(High packet rate)



Data Path Delay – Mean / std dev tables

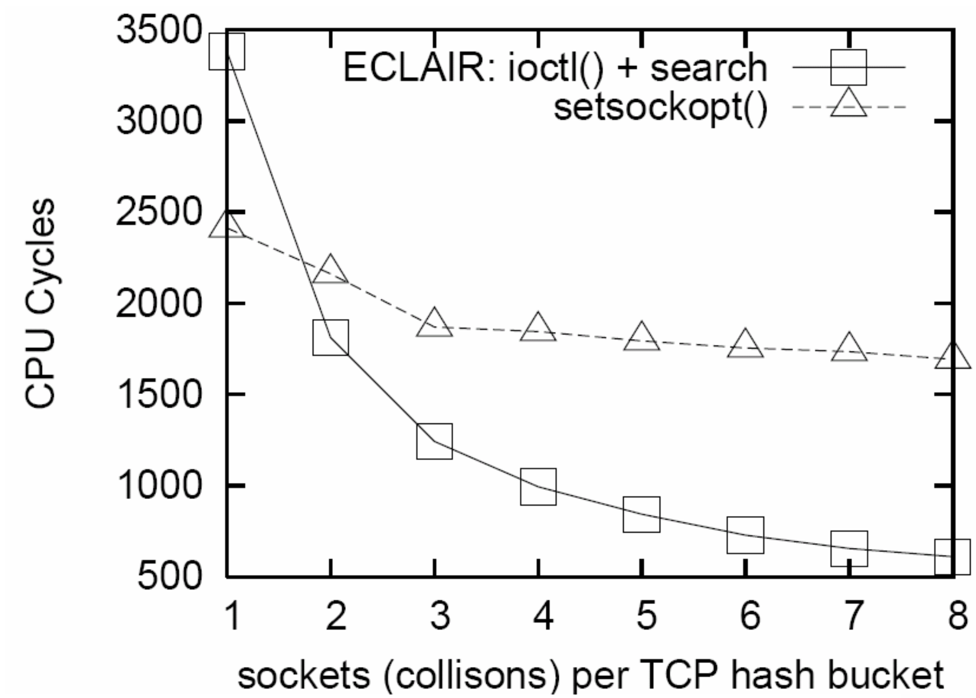
Invoke frequency (per second)	Load $\times 10^5$	$\bar{X} \times 10^{-5}$	$\sigma \times 10^{-5}$
No CLF		4.810 ± 1.171	19·847
10	2	4.903 ± 1.859	30·417
	5	4.411 ± 1.103	18·507
	10	4.506 ± 0.689	11·674
	15	12.161 ± 8.625	143·541
	20	7.884 ± 3.245	49·442
13	2	6.756 ± 3.076	51·198
	5	3.440 ± 0.175	2·956
	10	5.249 ± 1.684	28·147
	15	5.354 ± 0.854	14·326
	20	6.023 ± 1.634	26·828
20	2	3.489 ± 0.129	2·146
	5	3.882 ± 0.749	12·543
	10	6.093 ± 1.849	30·815
	15	7.038 ± 1.994	33·394
	20	9.630 ± 3.044	50·453
33	2	3.845 ± 1.067	16·908
	5	5.878 ± 3.053	47·806
	10	7.478 ± 2.384	40·349
	15	11.056 ± 3.320	54·697
	20	10.585 ± 1.916	31·925
100	2	4.905 ± 2.494	41·821
	5	6.179 ± 1.541	25·148
	10	11.309 ± 1.757	29·629
	15	18.495 ± 2.676	44·471
	20	28.501 ± 3.417	56·278
Synchronous (in TCP)	0.2	7.164 ± 2.853	48·296
	2	61.179 ± 39.668	699·929
	5	74.501 ± 4.334	72·778
	10	158.373 ± 6.292	105·015

Low packet arrival rate

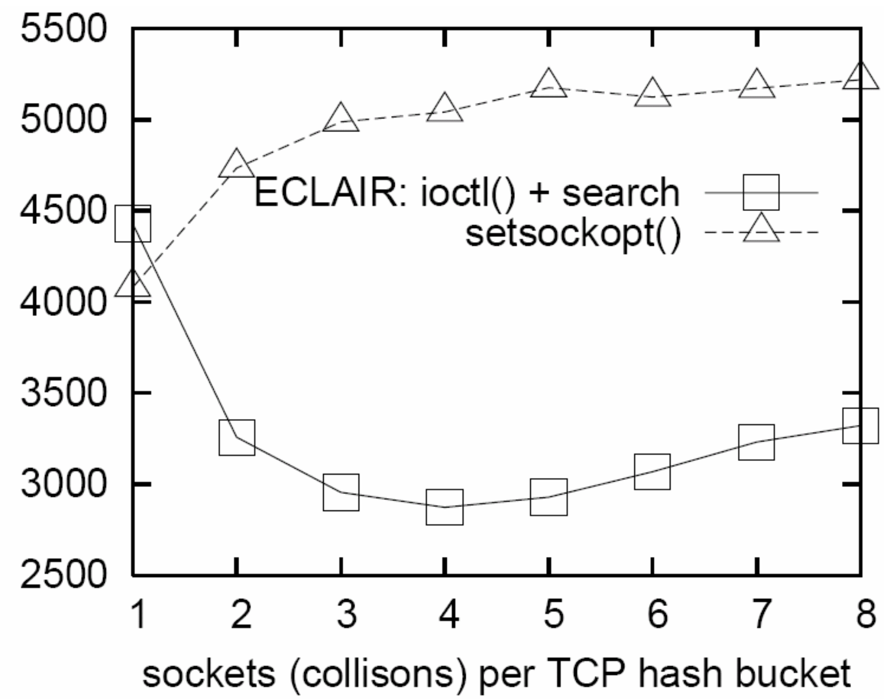
Invoke frequency (per second)	Load $\times 10^5$	$\bar{X} \times 10^{-5}$	$\sigma \times 10^{-5}$
No CLF		9.549 ± 4.406	152·759
10	2	7.486 ± 3.939	118·642
	5	11.542 ± 5.181	168·092
	10	44.305 ± 18.059	579·235
	15	38.024 ± 19.379	625·805
	20	28.674 ± 10.518	345·489
13	2	18.468 ± 8.868	285·295
	5	18.436 ± 8.301	265·439
	10	16.443 ± 6.980	225·76
	15	18.798 ± 7.506	242·977
	20	32.203 ± 15.980	523·711
20	2	12.966 ± 5.132	164·567
	5	24.851 ± 11.472	367·199
	10	18.599 ± 10.680	343·186
	15	17.993 ± 8.038	259·836
	20	36.171 ± 16.270	537·46
33	2	13.912 ± 6.766	219·902
	5	17.429 ± 8.934	287·137
	10	37.503 ± 17.780	571·269
	15	50.871 ± 21.883	731·215
	20	24.931 ± 11.905	399·365
100	2	29.779 ± 12.380	336·842
	5	32.297 ± 13.281	428·941
	10	23.471 ± 8.824	282·737
	15	44.926 ± 18.624	601·483
	20	39.796 ± 9.016	299·111
Synchronous (in TCP)	0.2	15.792 ± 10.137	318·303
	2	75.832 ± 13.294	501·423
	5	125.026 ± 9.725	338·057
	10	217.152 ± 10.074	351·64

High packet arrival rate

User-Kernel Crossing + search



CPU caching allowed
Array used



CPU cache invalidated
Array used

User-kernel crossing Mean/std dev tables

Sockets per bucket	Mean	Std dev
1	10655 ± 82	512
2	10850 ± 62	549
3	11107 ± 58	635
4	11285 ± 113	1422
5	11426 ± 55	776
6	11642 ± 51	780
7	11815 ± 98	1628
8	11823 ± 52	921

ioctl(), NO array, CPU caching

Sockets per bucket	Mean	Std dev
1	2898 ± 289	1809
2	2242 ± 165	1460
3	2032 ± 116	1262
4	1918 ± 87	1090
5	1834 ± 70	981
6	1789 ± 59	910
7	1753 ± 50	839
8	1729 ± 44	791

setsockopt, NO array, CPU caching

Sockets per bucket	Mean	Std dev
1	10158 ± 86	537
2	10456 ± 65	577
3	10538 ± 58	633
4	10700 ± 53	667
5	10946 ± 124	1734
6	11196 ± 52	799
7	11315 ± 103	1718
8	11525 ± 73	1290

ioctl(), NO array, CPU cache invalidated

Sockets per bucket	Mean	Std dev
1	4633 ± 159	996
2	5019 ± 93	826
3	5168 ± 71	773
4	5156 ± 56	711
5	5225 ± 52	732
6	5268 ± 49	753
7	5332 ± 93	1542
8	5278 ± 38	680

setsockopt(), NO array, CPU cache invalidated

User-kernel crossing Mean/std dev tables

Sockets per bucket	Mean	Std dev
1	2535 ± 549	3434
2	1365 ± 310	2744
3	952 ± 212	2296
4	767 ± 168	2039
5	656 ± 136	1901
6	570 ± 112	1726
7	522 ± 98	1635
8	486 ± 88	1539

ioctl(), Array, CPU caching

Sockets per bucket	Mean	Std dev
1	2274 ± 181	1133
2	2014 ± 121	1069
3	1824 ± 81	877
4	1772 ± 66	801
5	1738 ± 52	736
6	1702 ± 44	677
7	1685 ± 39	658
8	1654 ± 33	583

setsockopt, Array, CPU caching

Sockets per bucket	Mean	Std dev
1	3735 ± 424	2654
2	2902 ± 247	2184
3	2713 ± 170	1841
4	2712 ± 130	1630
5	2828 ± 106	1487
6	2980 ± 94	1440
7	3153 ± 85	1408
8	3286 ± 82	1456

ioctl(), Array, CPU cache invalidated

Sockets per bucket	Mean	Std dev
1	4345 ± 145	906
2	4922 ± 90	801
3	5092 ± 68	745
4	5135 ± 61	768
5	5277 ± 186	2608
6	5173 ± 42	648
7	5213 ± 39	649
8	5254 ± 41	726

setsockopt(), Array, CPU cache invalidated

Results

- Modification to protocol stack can significantly increase data path delay
  – Applies to architectures such ISP, MobileMan, CLASS, ICMP Messages
- ECLAIR impact on data path is much lower, compared to protocol stack modification
  – At a load of 10 x 10^5 , invoke frequency of 100 times per second
    - ECLAIR : ~3 times of No CLF
    - Modification to protocol: ~30 times of No CLF
- Appropriate design using ECLAIR can help reduce user-kernel crossing overhead

Qualitative Evaluation
Key Architectural Features and Impact

Feature	Efficiency	Maintainability	Architectures
Components outside stack/within kernel	Low data path delay High time/space overhead	High	PMI, ICMP Messages, ECLAIR
Integrated within stack	High data path delay Low time/space overhead	Low	MobileMan, ISP, CLASS, ICMP Messages
Components in user-space	Low-high data path delay High user-kernel crossing	High	User-space. Certain extent PMI, ICMP, ECLAIR

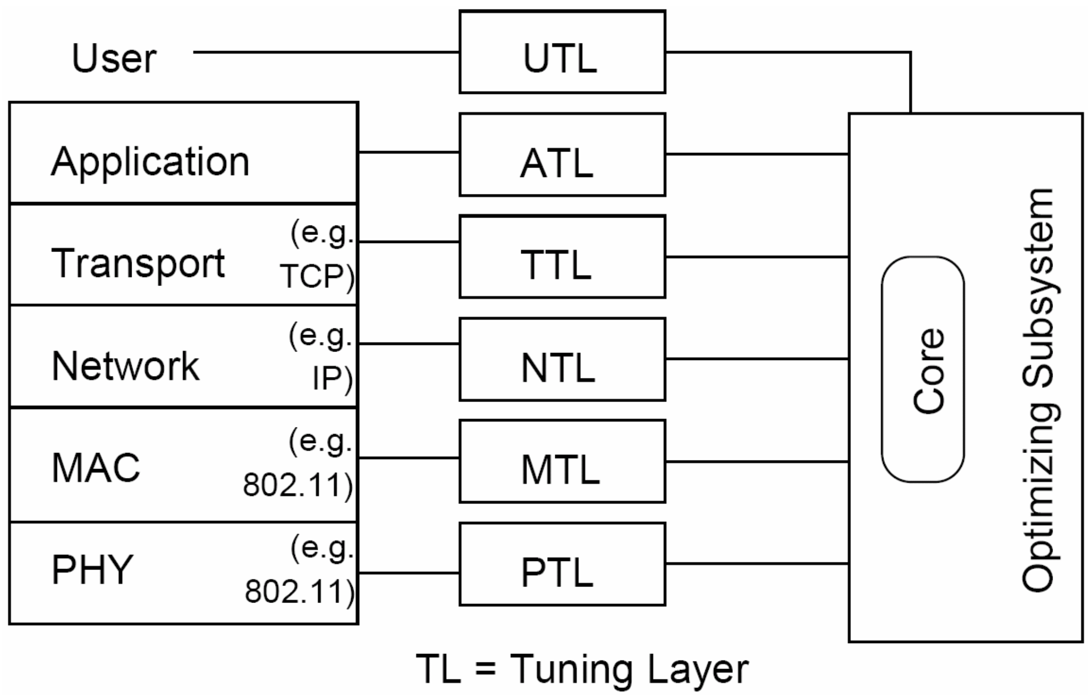
- Any-to-any layer cross layer feedback supported by ECLAIR, CLASS, MobileMan

ECLAIR Optimization

- To maximize benefit from cross layer feedback
  - Identify critical data items
  - Minimize overhead
- Critical data items
  - Provide high utility – improvement in stack efficiency
- Partition critical data items into two sets
  - Partition based on cost of cross layer feedback
  - Cost of cross layer feedback would reduce when an item is placed in core

ECLAIR Optimization: Core

- Core: Set of data items picked from layers; separately cached to reduce ECLAIR overhead
- Data item selection: Choose data items offering high utility; consider cost of read/write from/to core v/s non-core



Core Item Selection

- Item suitable for core, if core potential score i.e. increase in efficiency > 0

$$1-\frac{c_r}{\bar{c}_r}-\frac{c_w}{\bar{c}_r}\times\frac{\omega_i'}{\omega_i}>0$$

c_w = cost of single write of item into core

c_r = cost of single read of item from core

$\bar{c}_r$ = cost of single read of item not in core

ω_i' = sum of estimated frequency of writing into core

ω_i = sum of estimated frequency of frequency of access by all layers other than generating layer

Identifying Critical Data Items

- Utility of data item
  - Frequency of access by layers other than layer generating cross layer item
  - d_i at layer j. ω_i = sum of frequency of access by layers $i \neq j$
- Order items by ω, select items above threshold

$$D = \{d_i : \omega_i > v\}$$

Core: Cost of Data Item

- Cost related to data item
  - Writing into core $\phi_i' = c_w \times \omega_i'$
  - Reading from core $\phi_i = c_r \times \omega_i$

c_w = cost of single write of item into core

c_r = cost of single read of item from core

$\bar{c}_r$ = cost of single read of item not in core

ω_i' = sum of estimated frequency of writing into core

ω_i = sum of estimated frequency of frequency of access

by all layers other than generating layer

Core: Costs

- Core interaction cost $\Psi = \sum(\phi_i' + \phi_i)$
- Total utility of core $\Theta = \sum \omega_i$
- Item suitable for core, if core potential score i.e. increase in efficiency > 0

$$(\bar{c}_r \times \omega_i) - (\phi_i' + \phi_i) > 0$$

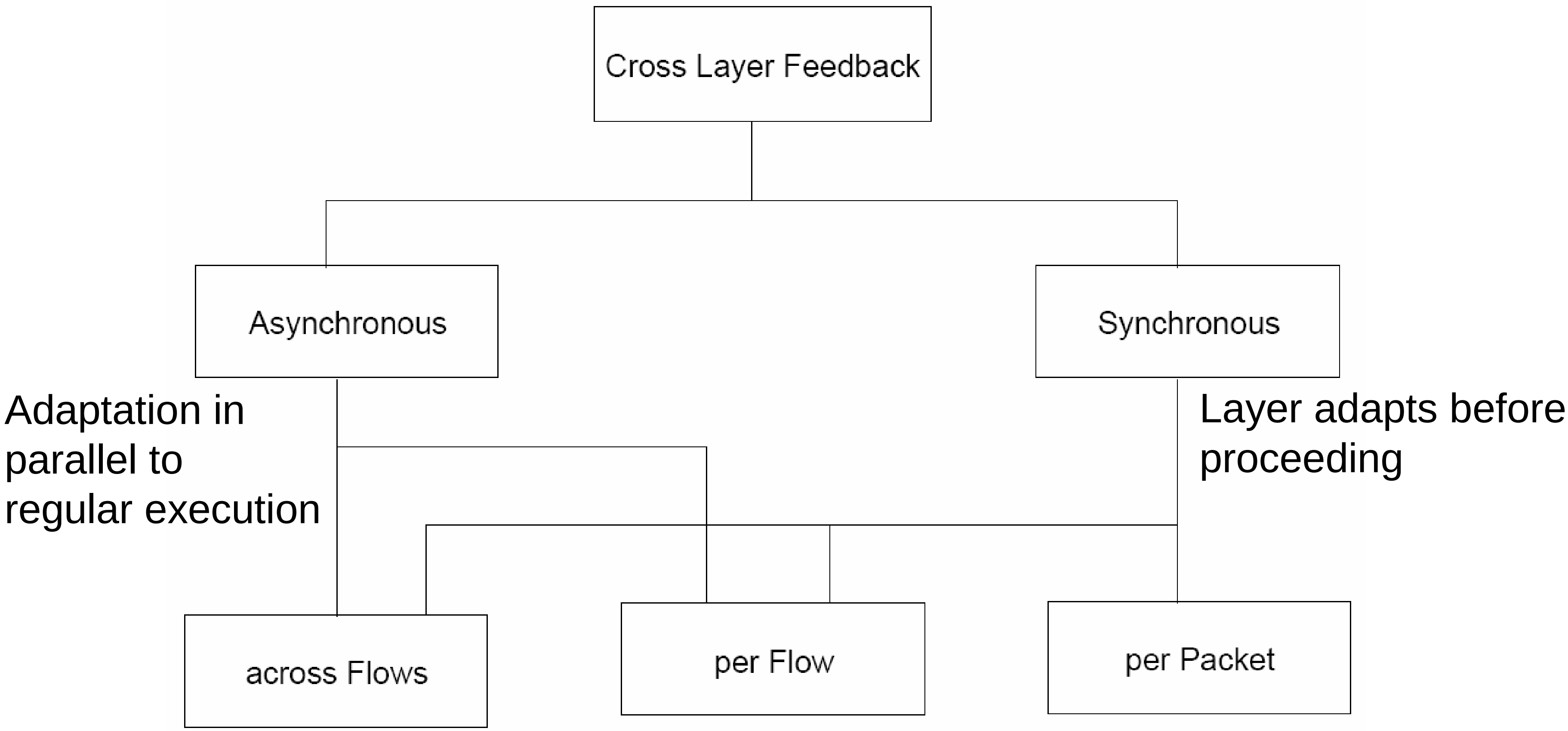
$$1 - \frac{c_r}{\bar{c}_r} - \frac{c_w}{\bar{c}_r} \times \frac{\omega_i'}{\omega_i} > 0$$

Core Item Selection

- Sort items on their Core Potential Score (descending)
- Select items till *utility of core* less *cost of core* is higher than specified design threshold

$$
\begin{array}{l}
\text{for all } d_i \in D' \text{ do} \\
\quad \text{if } \Theta - \Psi < \tau \text{ then} \\
\qquad C = C \cup \{d_i\} \\
\quad \text{else} \\
\qquad \text{break} \\
\quad \text{end if} \\
\text{end for}
\end{array}
$$

Cross Layer Feedback Types



Flow is a connection established over a path over the nodes in network

Architecture Selection

- Impact on efficiency
  - Synchronous architecture for asynchronous requirement leads to increased data path delay
- Impact on correctness
  - Asynchronous architecture for synchronous requirement
    - Difficult to synchronize cross layer system with stack execution
    - If synchronized, could lead to increased data path delay, since not well integrated with stack
- ECLAIR suited for synchronous cross layer feedback, since outside stack

ECLAIR Limitations

- May require modification to stack, if some data structure not accessible
- Per packet adaptation not built-in
  - However can be provided
- Direct solution to problems intrinsic to cross layer feedback not provided
  - Cross layer conflict
  - Protocol correctness
  - However, components can be used for addressing this

Security Issues

- If ECLAIR allows interaction with the network, authentication mechanism may be required
- Certification/signing may be required to protect ECLAIR components from malicious attacks

Contributions

- ECLAIR: Architecture for CLF
  - Definition, prototype implementation, validation(RWC)
- Core: Sub-architecture for reducing overheads
- Metrics for CLF architecture evaluation
- Notation for layer and CLF implementation aspects
- Design guide for cross layer feedback

Directions for Future Work

- Improve synchronous cross layer feedback efficiency of ECLAIR
  - Optimizations to reduce data path delay
- Enhance ECLAIR sub-architecture
  - Determine exact read/write costs and models to determine utility
- Extend ECLAIR for base-station and other nodes
  - Components for device specific adaptation and identification of connections
  - Scaling to large number of connections
- Extend ECLAIR for seamless mobility
  - Network node component to interact with device and aid seamless mobility
- Enhance ECLAIR to resolve conflicts and dependency cycles
  - Special PO to collect information from TLs and detect cycles
  - Extend TLs to permit TL behavior change on the fly

Acknowledgment

- Advisor: Prof. Sridhar Iyer
- Employer: TATA Infotech Ltd. (Now TCS), Dr. Arun Pande (Head Adv. Tech and Apps, TCS)
- PhD Committee: Professors Abhay Karandikar, Anirudha Sahoo, Krishna Paul
- Colleagues at TCS and IIT
- My parents, wife and kids

Publications

All with Sridhar Iyer

- **Journal/Magazine**
  1. (1st review completed, Oct 2006) ECLAIR, Architecture Evaluation.
     IEEE Transactions on Mobile Computing
  2. ECLAIR overview, RWC Implementation.
     IEEE Communications, Jan 2006
  3. Cross Layer Survey.
     Computer Communications (Elsevier), May 2004
- **Conference**
  1. ECLAIR overview, RWC, evaluation, sub-architecture.
     IEEE/ACM COMSWARE, N.Delhi, Jan 2006.
  2. RWC Analysis.
     IEEE ICPWC, N.Delhi, Jan 2005
  3. ECLAIR overview.
     World Wireless Congress, SF, USA, May 2004
  4. User Feedback.
     23rd ICDCS, USA, 2003 (Poster)
  5. (with AK Singh and Sridhar Iyer) Benefits of cross layer feedback, Receiver Window Control, ATCP.
     IEEE ICPWC, N.Delhi, Dec. 2002.

Thank you