

BART

Bandwidth Available in Real-Time

Svante Ekelin

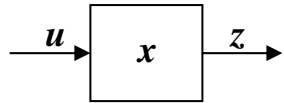
Network Control Lab, Ericsson Research (EAB)

Evergrow workshop, Såstaholm, December 2, 2005

EAB activity in EVERGROW 2005

We developed **BART**, a method for **estimation of time-dependent available bandwidth**.

BART uses a **filtering method** to produce an updated estimate for each sampling.



$$x_k = f(x_{k-1}) + w_k$$

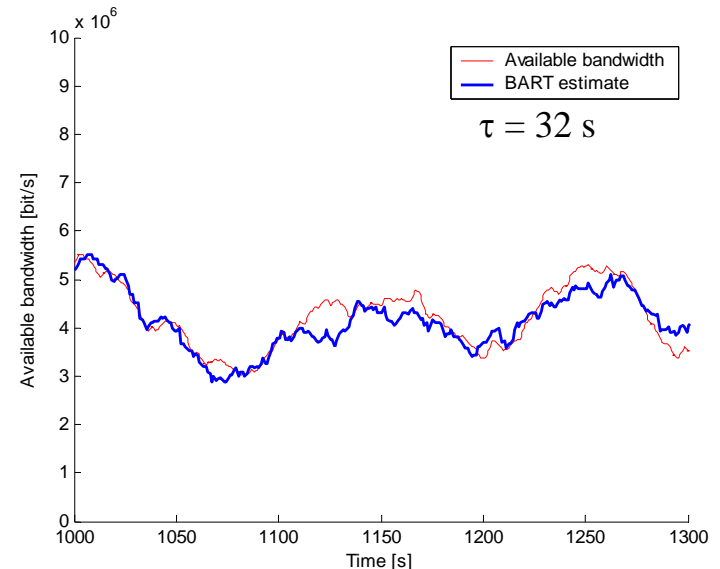
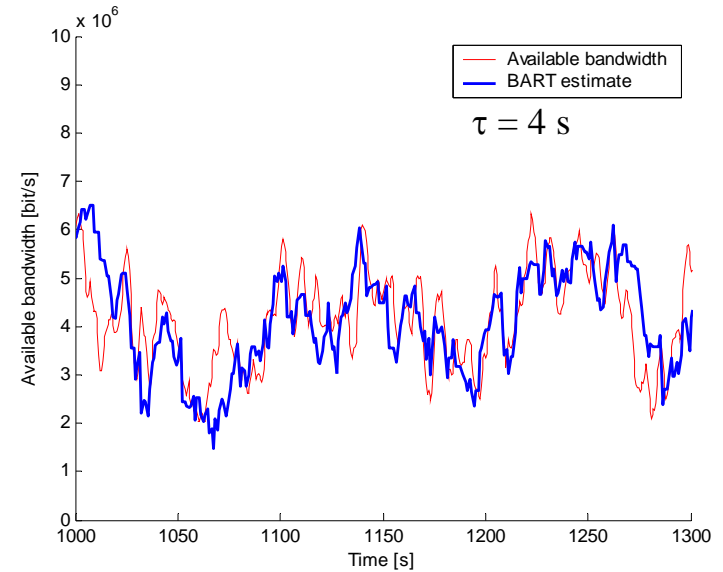
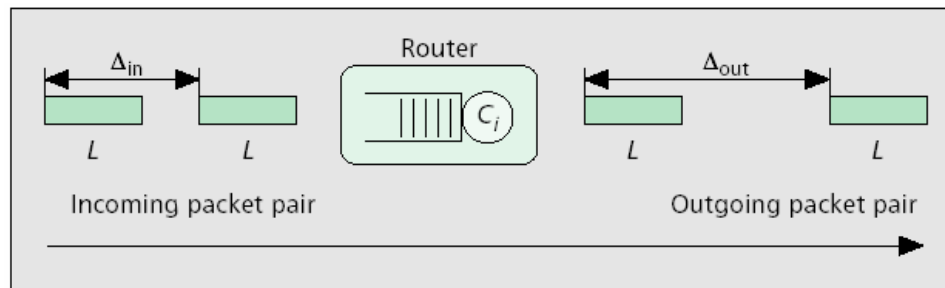
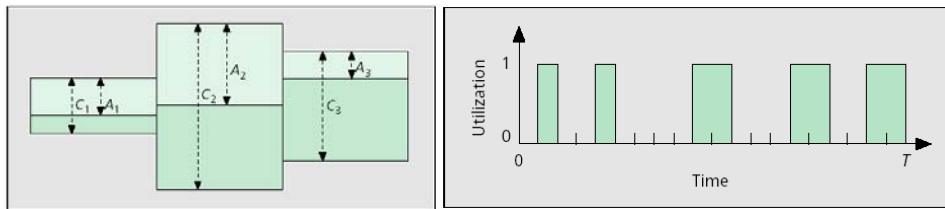
$$z_k = h(x_k) + v_k$$

Allows **recursive estimation** of system state:

$$\hat{x}_k = g(\hat{x}_{k-1}, z_k)$$

Benefits: **fast** (configurably)
accurate (reasonably)
tunable to desired time scale

Artefacts: **C++ implementation**
several papers

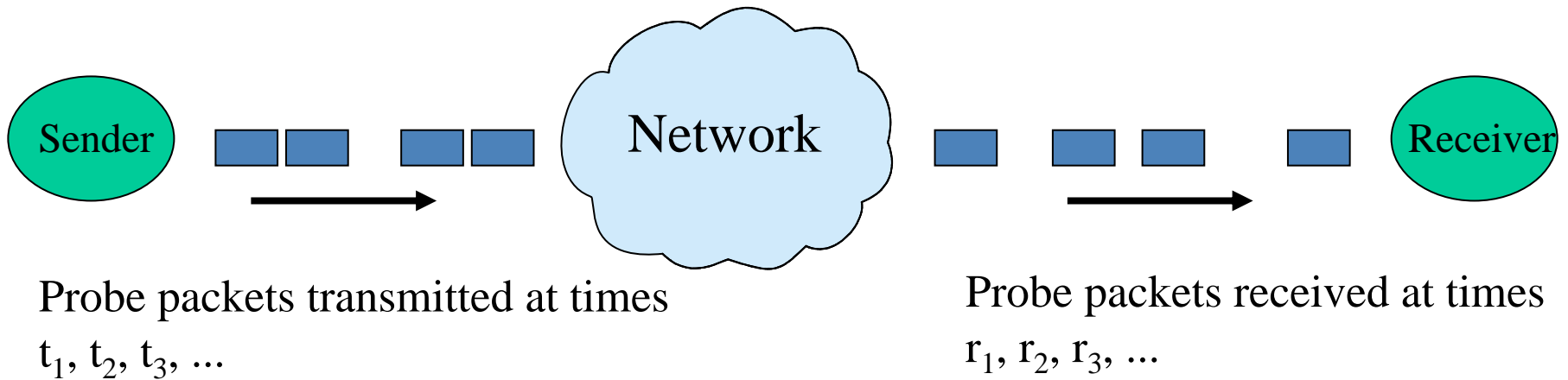


Probing the network

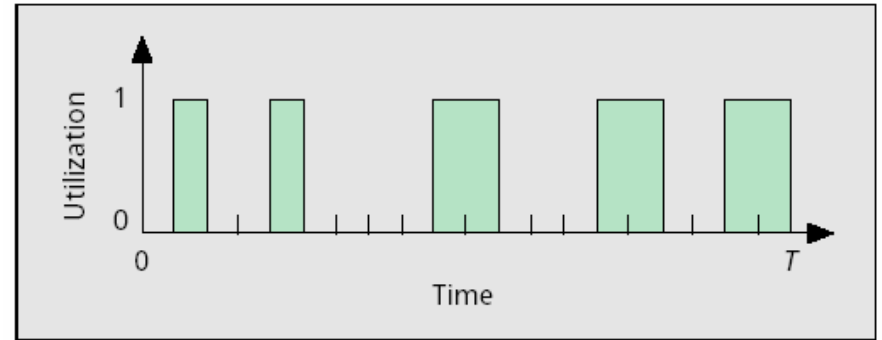
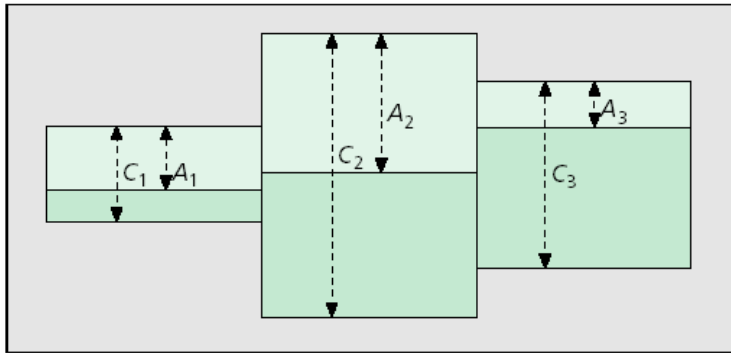
Basic idea behind active measurements:

1. Inject “probe packets” into the network
2. Observe the effect exercised on the probes by the network
3. Draw conclusions about the state of the network

(step 3 is the hard one...)



End-to-end available bandwidth



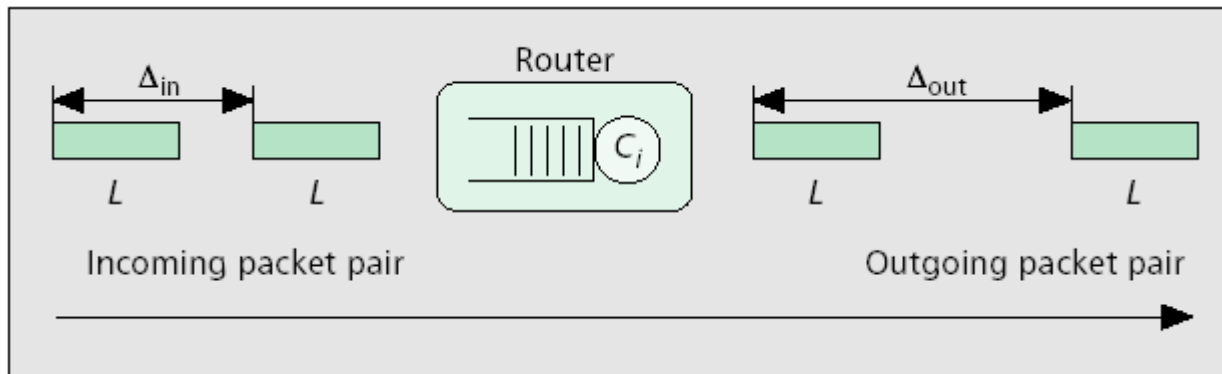
Hop number i along the path has capacity C_i and available bandwidth A_i . The available bandwidth of the path is the smallest of the link available bandwidths:

$$A = \min_i A_i.$$

[Note that traffic load and thus available bandwidth are defined relative to some arbitrary time resolution. Instantaneous utilization of a link is either 0 or 100% .]

Congestion => time dispersion

We probe the network path using pairs of probe packets.



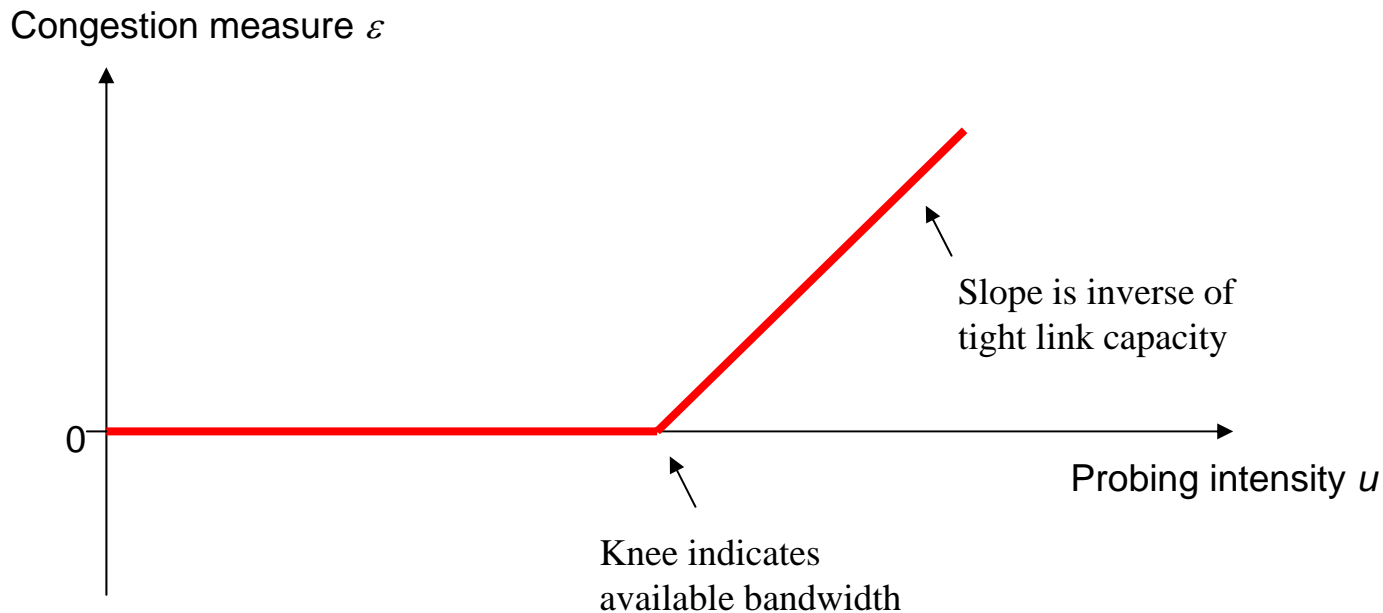
If the local probing rate $u = L / \Delta_{in}$ is larger than the available bandwidth, there is (transient) congestion.

As a congestion measure, we use the relative time dispersion, or "strain", $\varepsilon = (\Delta_{out} - \Delta_{in}) / \Delta_{in}$

$$[\varepsilon = u_{in} / u_{out} - 1]$$

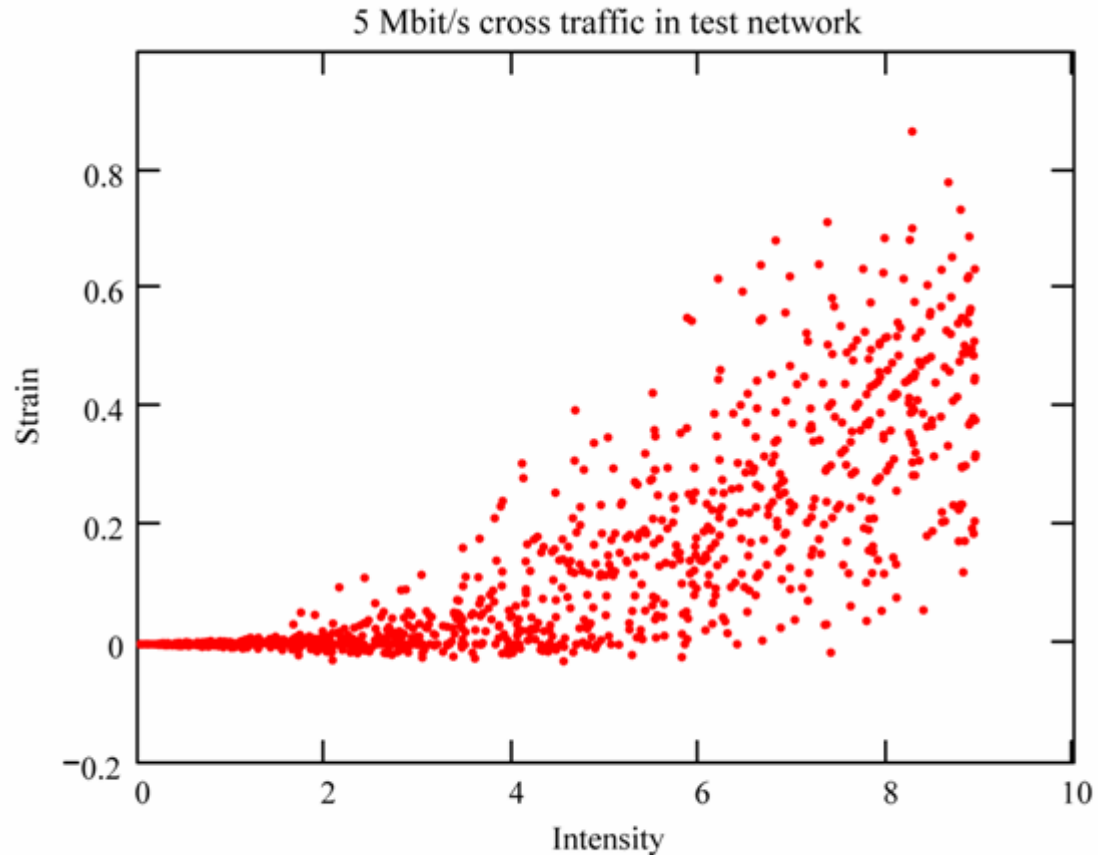
System curve

In a fluid model of traffic flow, one obtains a piecewise linear dependence of the strain on the probing intensity



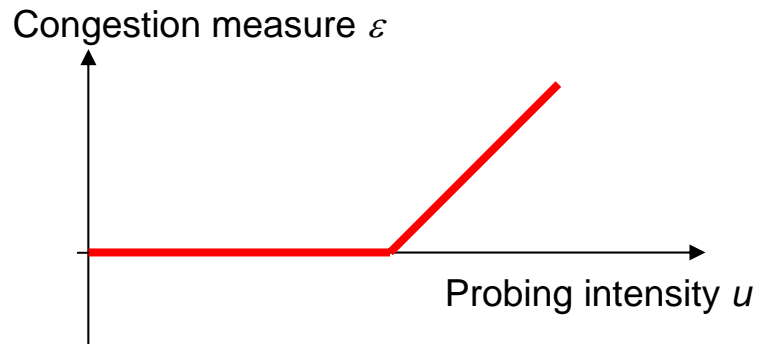
BUT, we have "horizontal noise" and "vertical noise"...

What we actually see...



Measured strain vs. probing intensity in a test network. Cross traffic is Poisson with average intensity 5 Mb/s . Tight link capacity is 10 Mb/s .

State-of-the-Art methods



Pathload

- Looks for increasing strain as signature of overload
- Probes with trains of varying intensity, with feedback from receiver to sender, to "fork in" the breakpoint
- Slow to produce an estimate (not "real-time")

TOPP

- Probes with trains of increasing intensity
- Fits two straight lines by linear regression
 - Difficult to do this since breakpoint is not known
- Slow to produce an estimate (not "real-time")

pathChirp

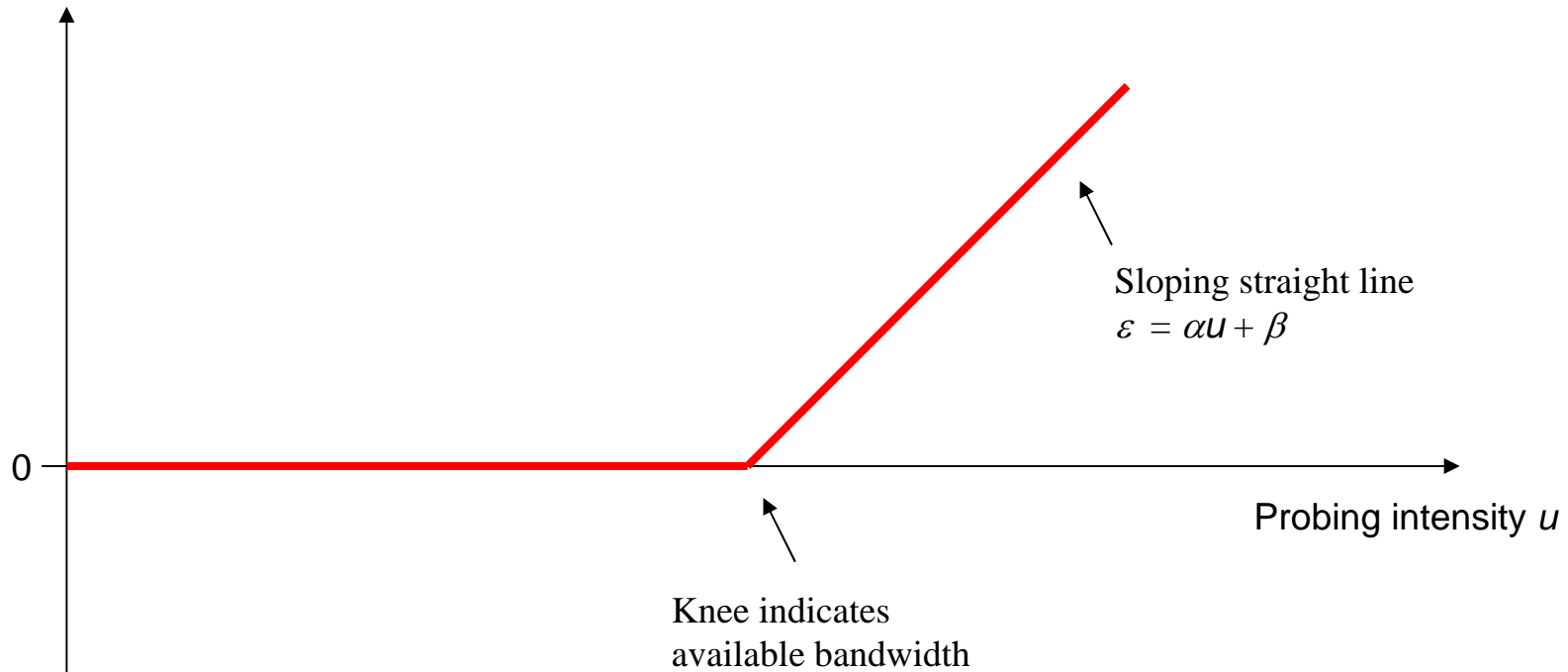
- Scans a range of probing intensities with each train (each "chirp")
- Looks for signature in delay pattern
- Improves on Pathload
 - Produces estimate for each chirp
 - No need for feedback to sender
 - Measurement precision not very high

BART

- Basic idea: use **filtering method** to produce an updated estimate for each sampling
- Recursive estimation:
 - new estimate = $g(\text{previous estimate}, \text{new measurement})$
- Benefits:
 - Fast like pathChirp
 - More accurate
 - Light-weight w.r.t. memory and CPU req's
 - Tunable
 - Open-loop (i.e. doesn't require feedback from receiver to sender)
 - Doesn't require synchronization between receiver and sender

system curve according to BART

measured strain ε

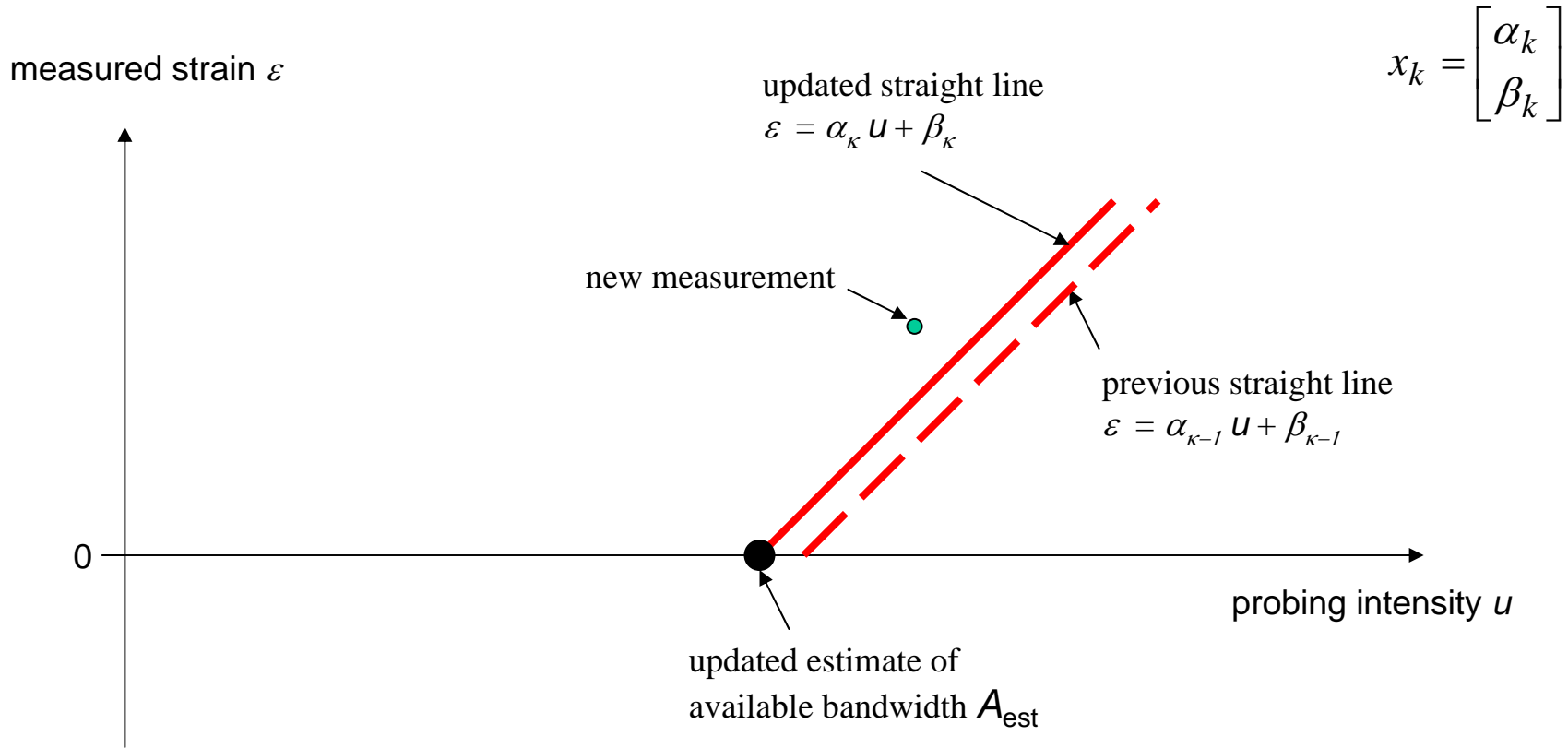


- System state described by parameters of sloping straight line : $x = \begin{bmatrix} \alpha \\ \beta \end{bmatrix}$
- System evolution described by the process $x_k = x_{k-1} + w_{k-1}$.
- Measurement as function of system state described by $\varepsilon_k = Hx_k + v_k$ where $H = \begin{bmatrix} u & 1 \end{bmatrix}$.
- w and v are the process noise and measurement noise. Their covariances: Q and R .

Kalman filtering

- This problem formulation allows using Kalman filtering, since
 - System state evolution is linear
 - Measured quantity is (approximately) linear function of system state
- Kalman filtering is
 - Well established and widely used (in other fields)
 - Light-weight
 - Provably optimal (under specific circumstances)
 - Even when these circumstances are broken, Kalman filtering generally produces good estimates.

Estimate update according to BART



- exactly how α and β are updated is given by the Kalman filter equations
- important input parameters: Q and R (covariances of process noise and measurement noise)
- R is measured by BART, whereas Q may be used for tuning (Q small \Rightarrow small change)

Kalman filter equations

"prediction"

"correction"

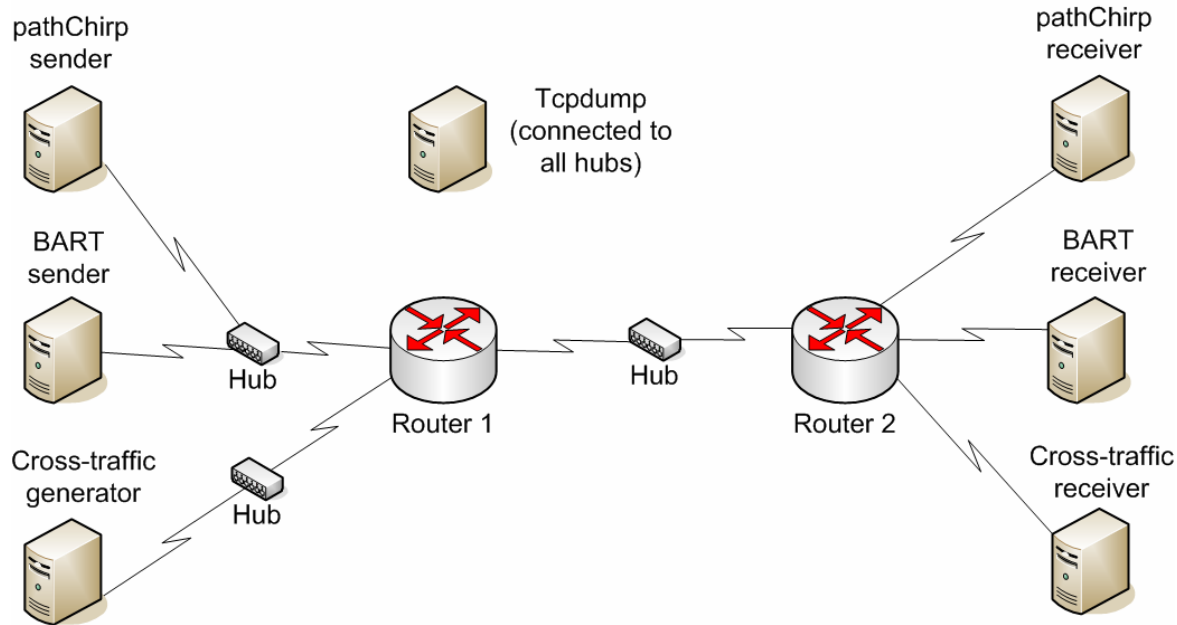
$$\hat{x}_k^- = A\hat{x}_{k-1}$$
$$P_k^- = AP_{k-1}A^T + Q$$

$$\hat{x}_k = \hat{x}_k^- + K_k(z_k - H\hat{x}_k^-)$$
$$P_k = (I - K_k H)P_k^-$$

where the Kalman gain $K_k = P_k^- H^T (HP_k^- H^T + R)^{-1}$

increases with Q and decreases with R

Empirical validation

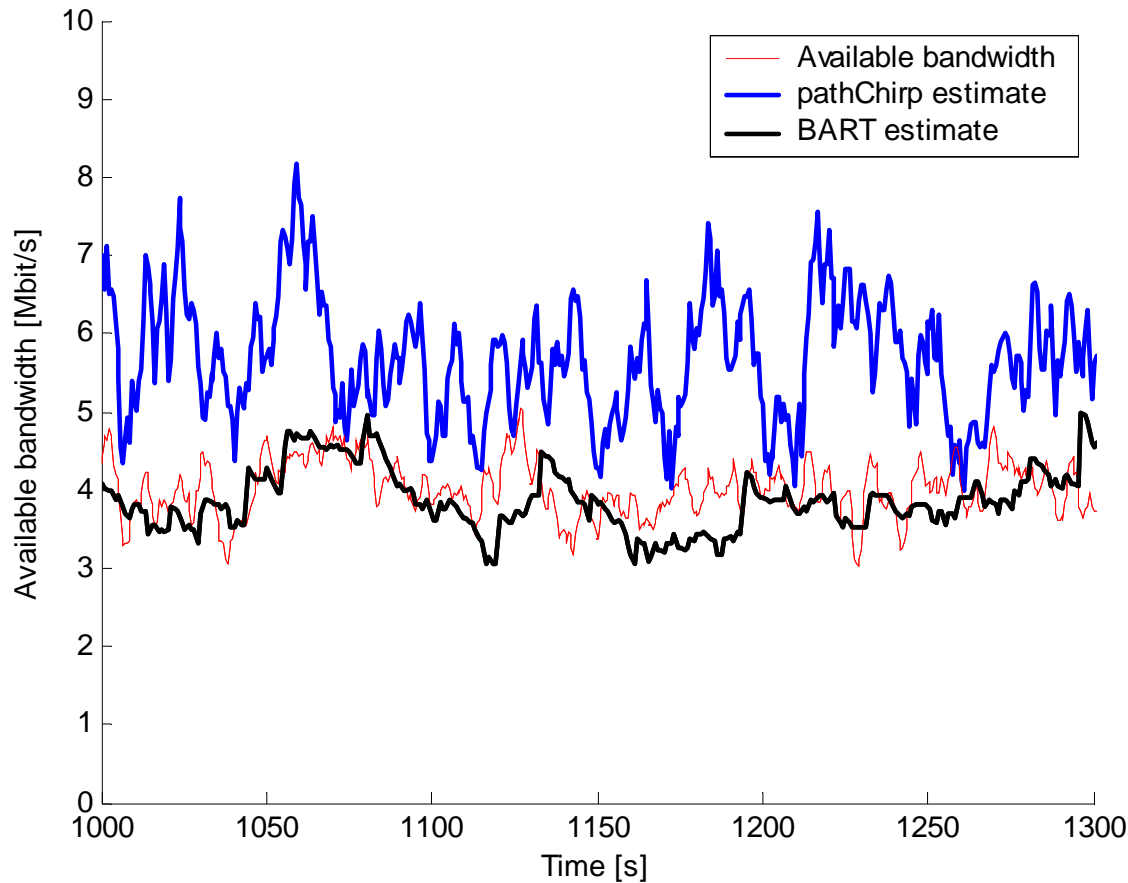


We are testing BART (a prototype C++ implementation running under Linux) and evaluating its performance in a controlled lab network environment.

The setup also allows comparing the performance of BART with that of pathChirp.

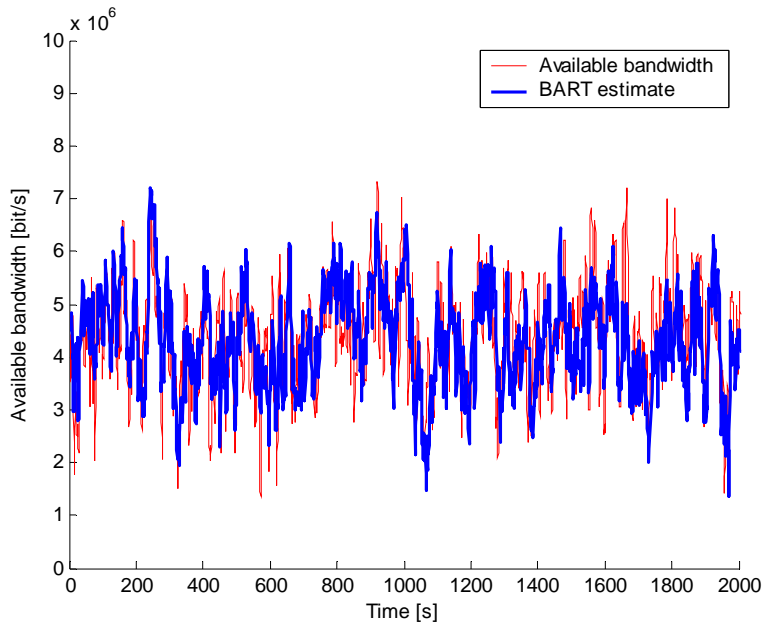
In order to be able to know the true available bandwidth, we trace all traffic with `tcpdump`.

Typical results

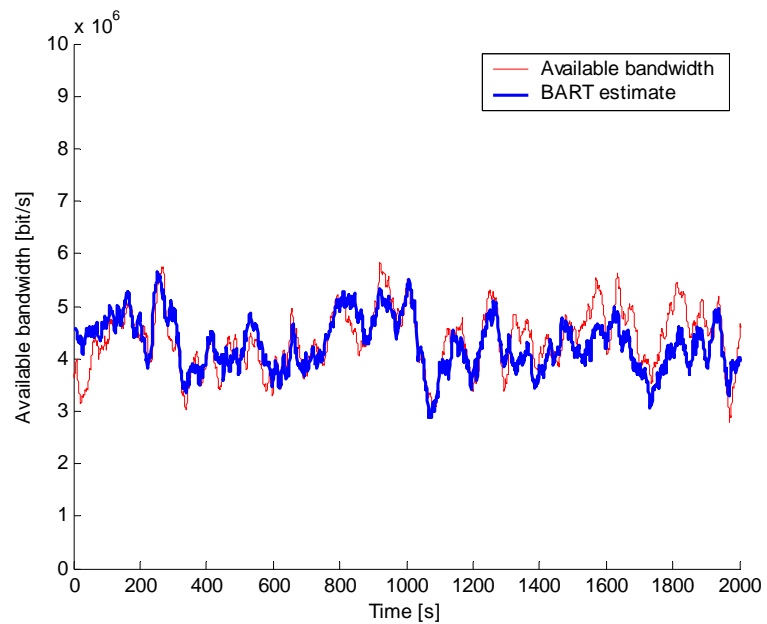


Aggregated traffic, ~100 users, Pareto characteristics

Tuning BART to desired time scale



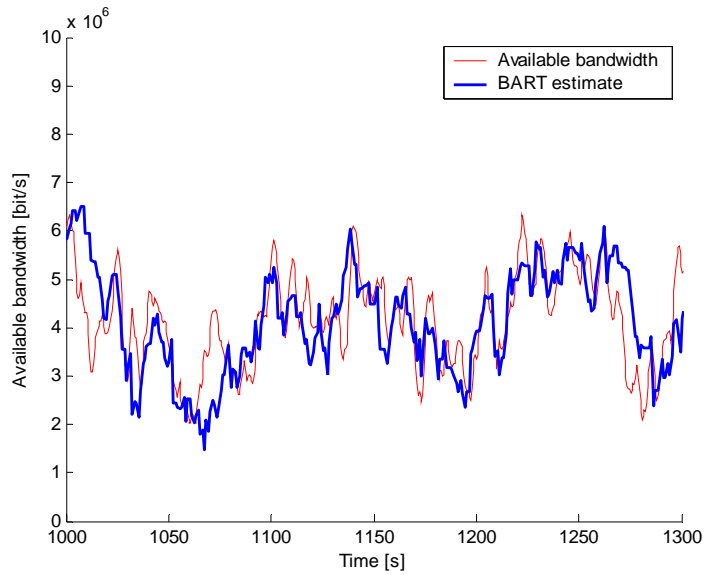
Averaging time scale: 4 s



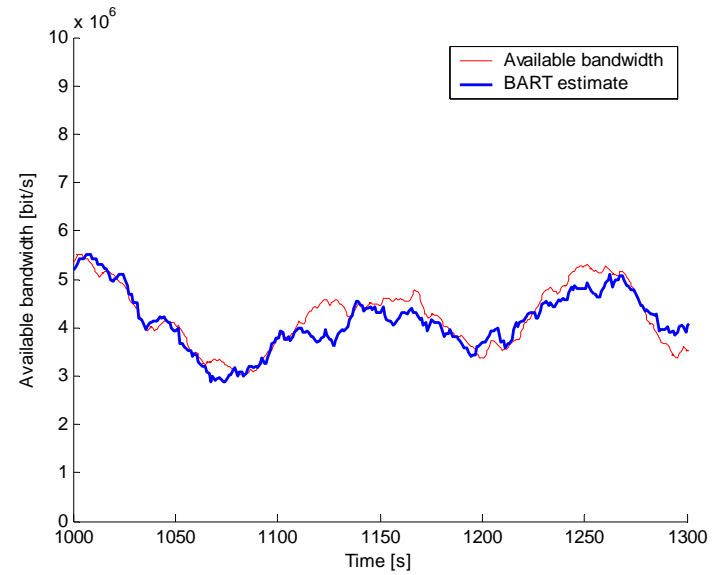
Averaging time scale: 32 s

Note that the true traffic is the same in both graphs.
(Aggregated traffic, ~ 10 users, Pareto characteristics)

zooming in...



Averaging time scale: 4 s



Averaging time scale: 32 s

Current status

- Continued performance enhancement work
- C++ implementation for Linux
- Plans to do large-scale experiments using DIMES agents as BART senders and ETOMIC boxes as receivers