

The Measurement Manager

Modular End-to-End Measurement Services

Ph.D. Research Proposal
Department of Electrical and Computer Engineering
University of Maryland, College Park, MD

Pavlos Papageorgiou

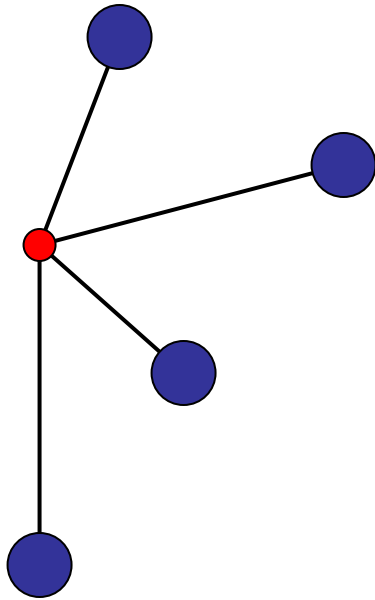
pavlos@eng.umd.edu

Network Measurement is Needed

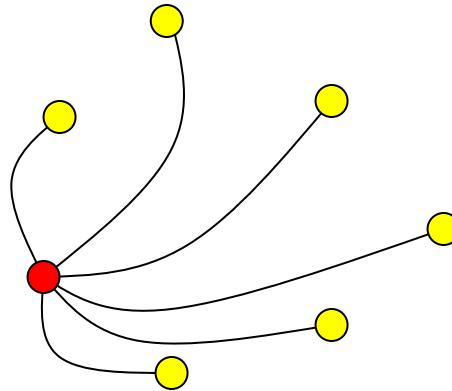
- **Applications** need to monitor end-to-end path
 - Due to the design of the Internet network routers do not provide any feedback about network conditions
 - Applications need to adapt based on path conditions
- **Overlays** need to select paths
 - Based on capacity, available bandwidth, delay, loss
- **Network services** need to select servers
 - Download managers need to select the best server
 - BitTorrent needs to select peers to download/upload

Network Measurement

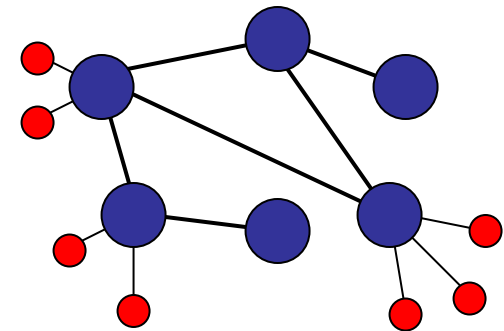
Server Selection Problem



BitTorrent: Peer-to-Peer



Overlays: MediaNet



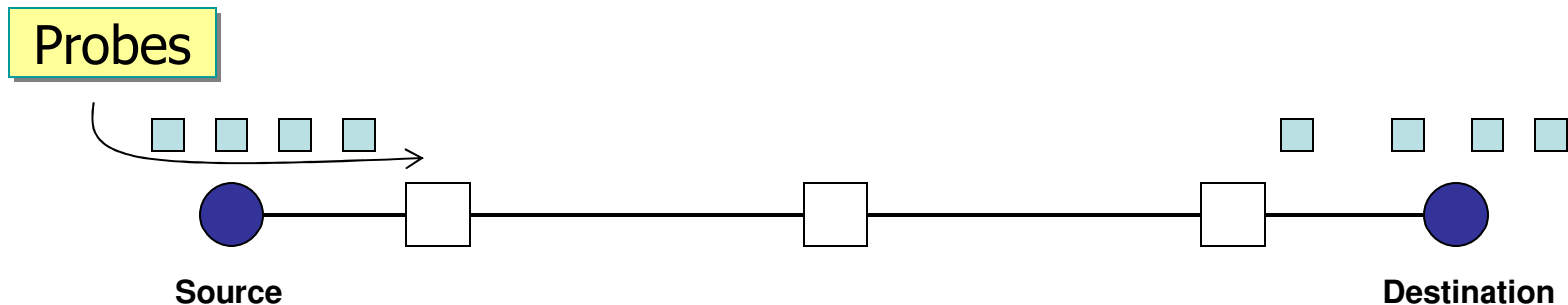
- Everyone is measuring the network

The Problem

- Network Measurement is inefficient, inflexible, and does not scale

Active Measurement

- What is it?
 - Injects special packets (probes) into the network
 - Sends packets in groups with accurate inter-packet gaps
 - Uses estimation algorithms to deduce network properties
 - Can be fast, accurate
 - Many standalone active measurement tools
- The Problem
 - **Not efficient:** probes carry empty padding
 - **Intrusive:** probes interfere with packets on the path we are measuring
 - **Usability:** not clear how standalone tools can be integrated
 - **Does not scale**



Passive Measurement

- What is it?
 - Observes existing packets and applies estimation algorithm
 - Typically tightly coupled with each application
 - TCP RTT estimation
 - RTP delay monitoring
 - Efficient, no overhead
- The Problem
 - **Unpredictable:** cannot generate packets when no packets available
 - **Inflexible:** no control over the packet sizes and packet gaps
 - **Inconsistent:** accuracy cannot be controlled

Our Goal

Enable transport protocols and applications to take advantage of **network measurement** as a service with **tunable overhead** and in a **modular way** by combining probes with transport packets.

Our Approach

A new Measurement Manager Architecture

Coordinate all measurement between two end-hosts
Combine best properties of Active and Passive approaches

Proposal Objectives

Motivation

Why do we need the measurement manager?

Architecture

Step-by-step example to present the architecture

Current Implementation & Preliminary Results

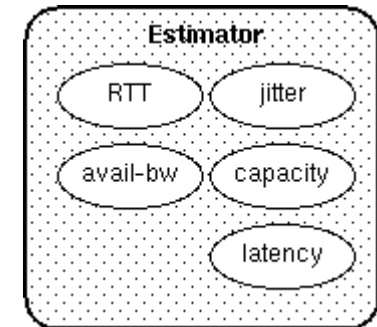
Proposed Work

Divided into three phases

The Measurement Manager

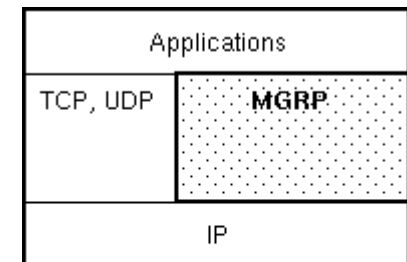
- **Estimator service**

- Network Measurement Service
- Collection of estimation algorithms
- Provides high-level interface to clients

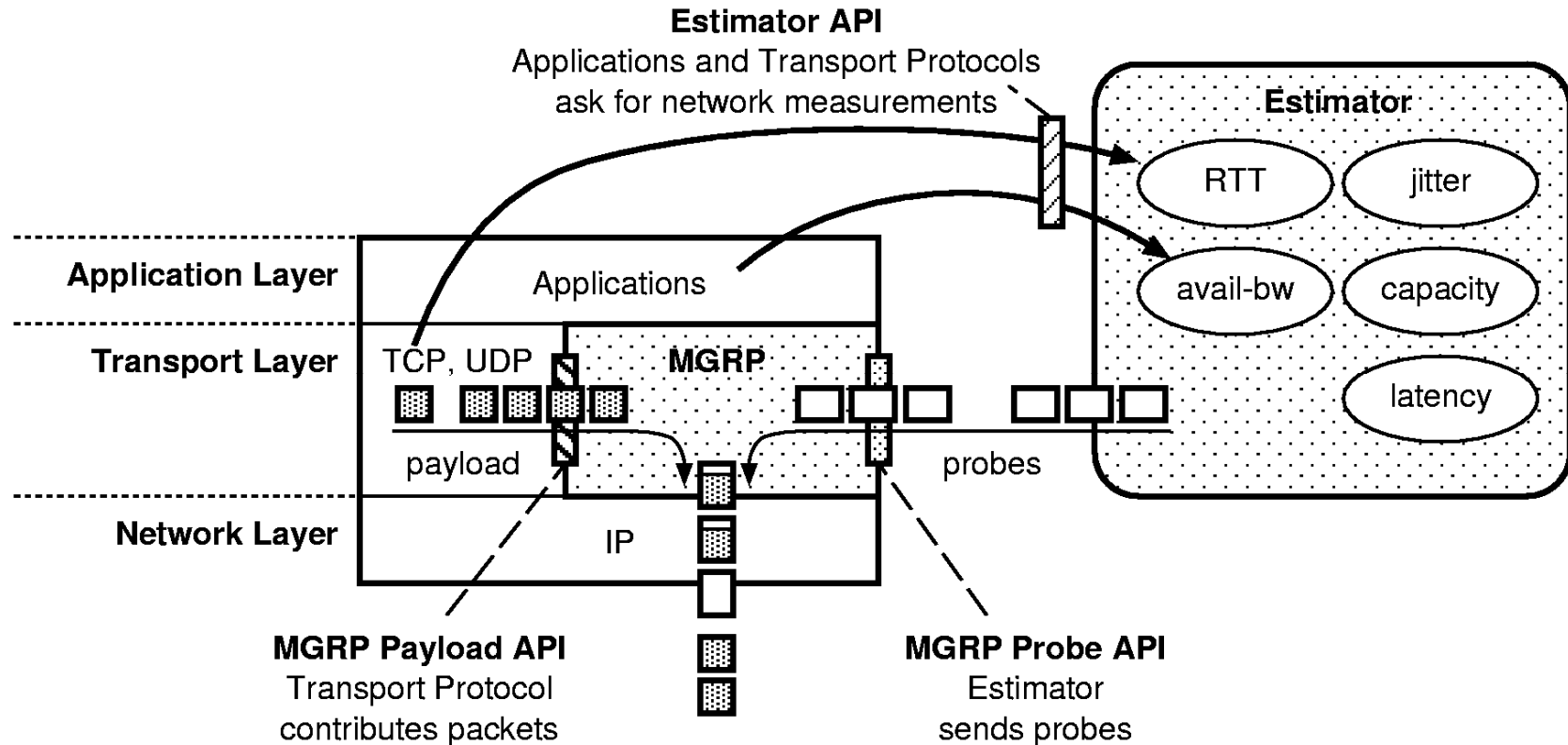


- **Probe scheduling service**

- MGRP: Measurement Manager Protocol
- New Layer-4 protocol
- Sends the probes efficiently by piggybacking on transport payload



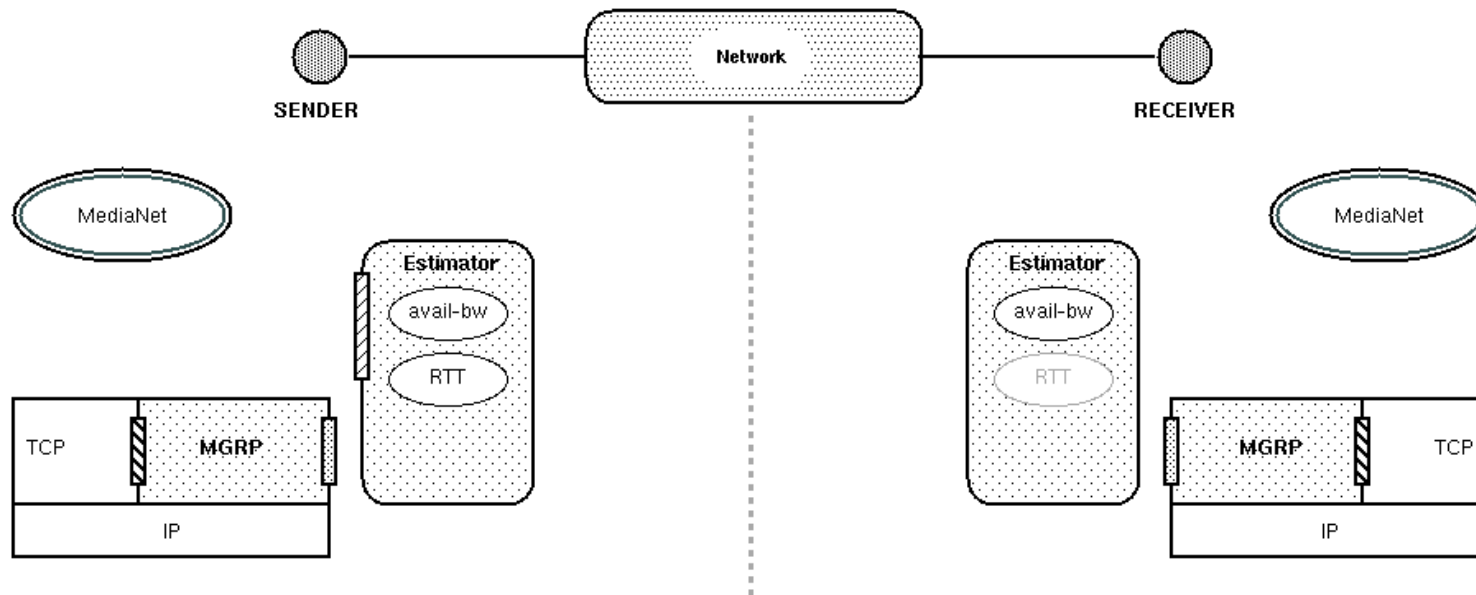
The Measurement Manager



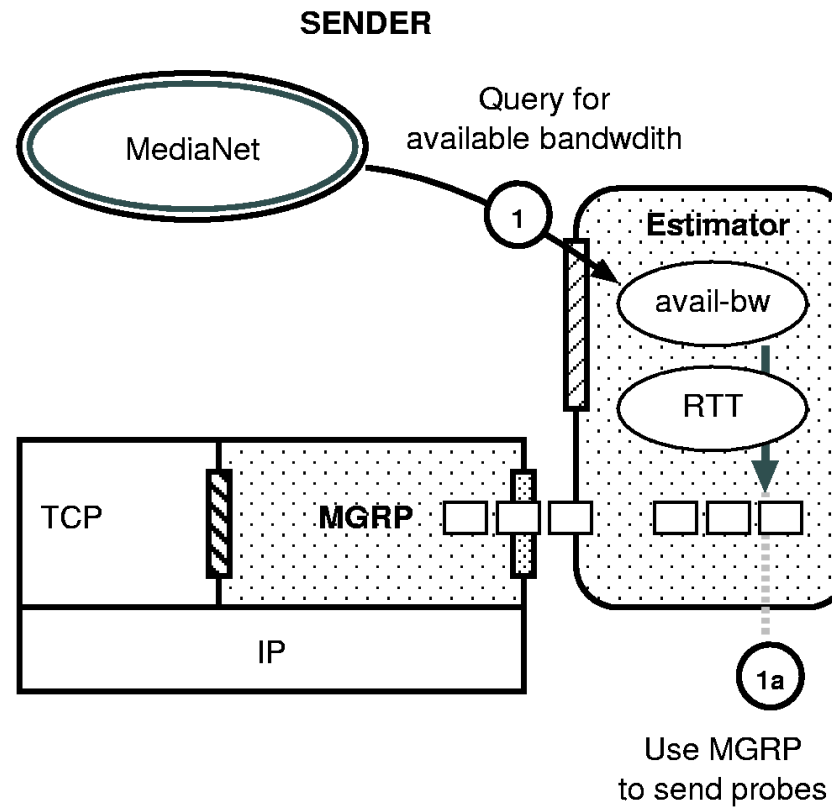
Novelty of our approach

- **Efficiency**
 - Uses transport payload to reduce the probing overhead
- **Flexibility**
 - Independent of applications, transport protocols
 - Each node can implement their own estimation algorithms independently using *probing primitives*
 - Estimation algorithms can be designed as if they are always active
- **Deployment path**
 - Probing and estimation can evolve separately

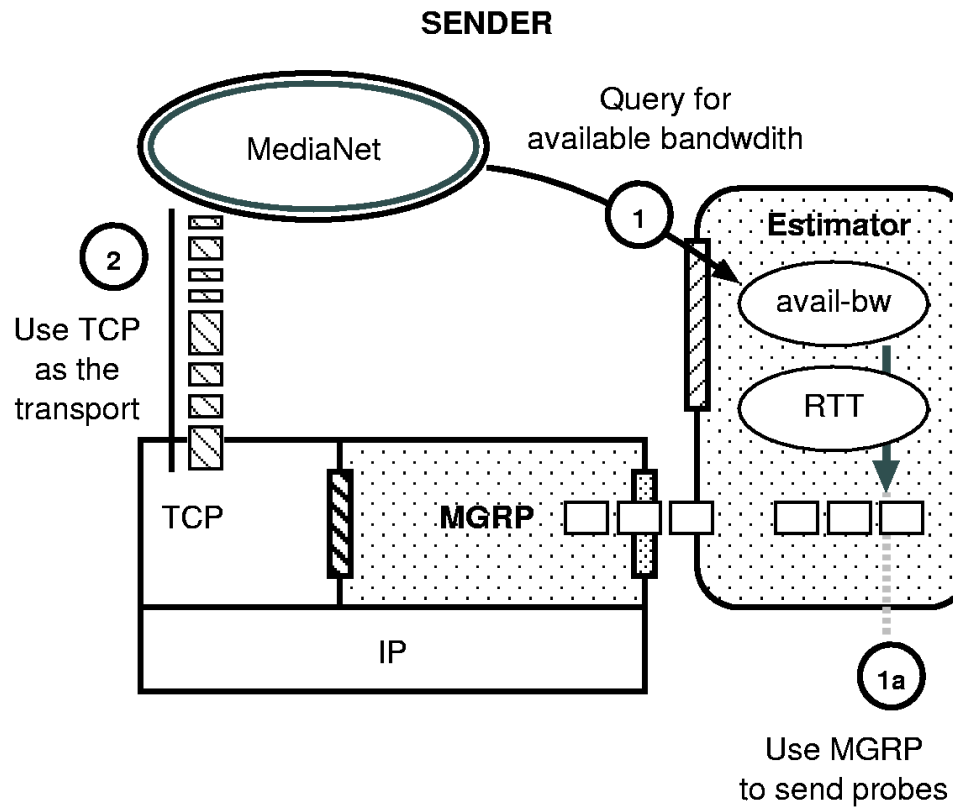
Architecture: Step-by-Step Example



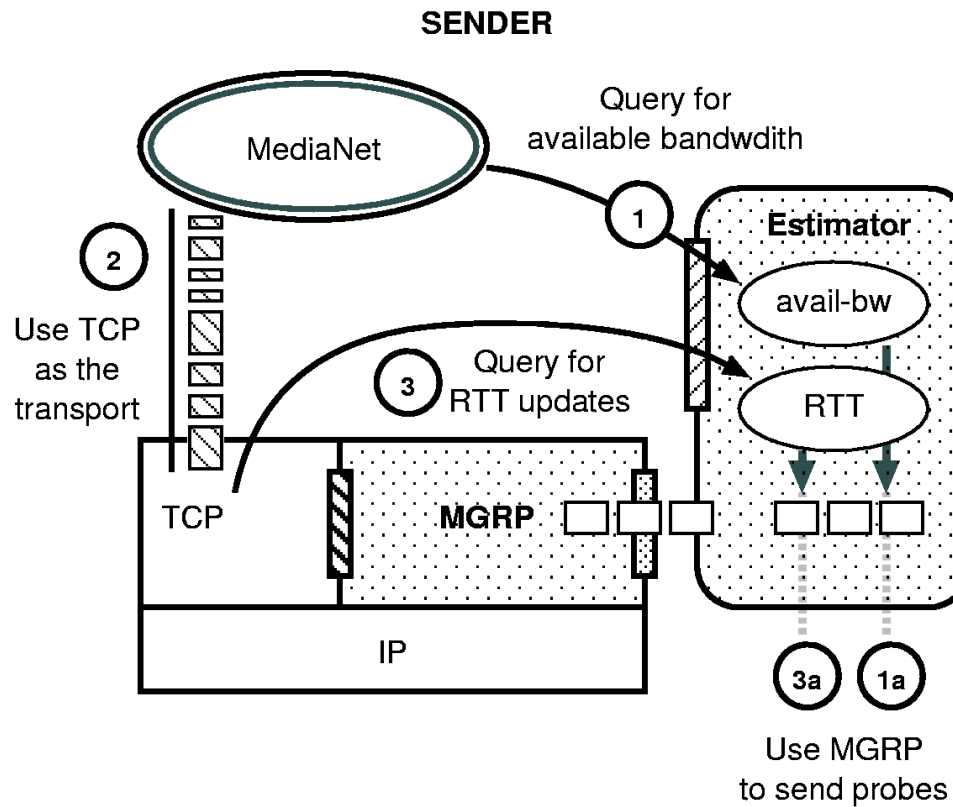
Architecture: Step-by-Step Example



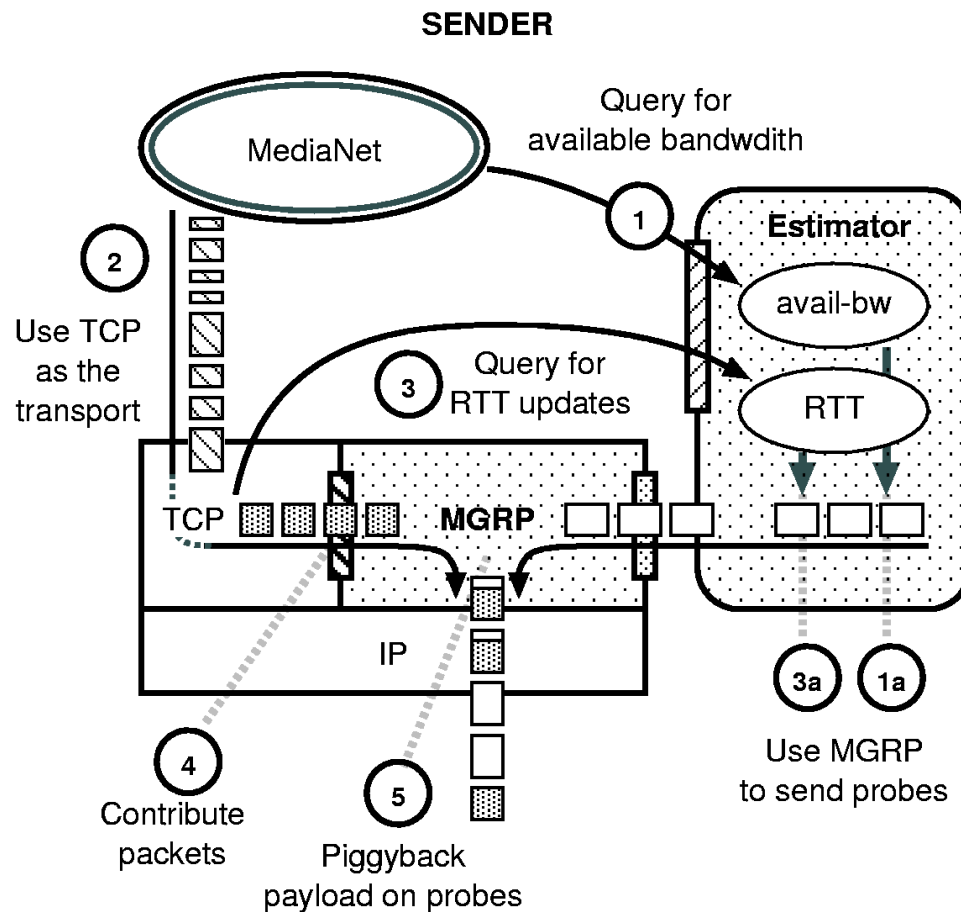
Architecture: Step-by-Step Example



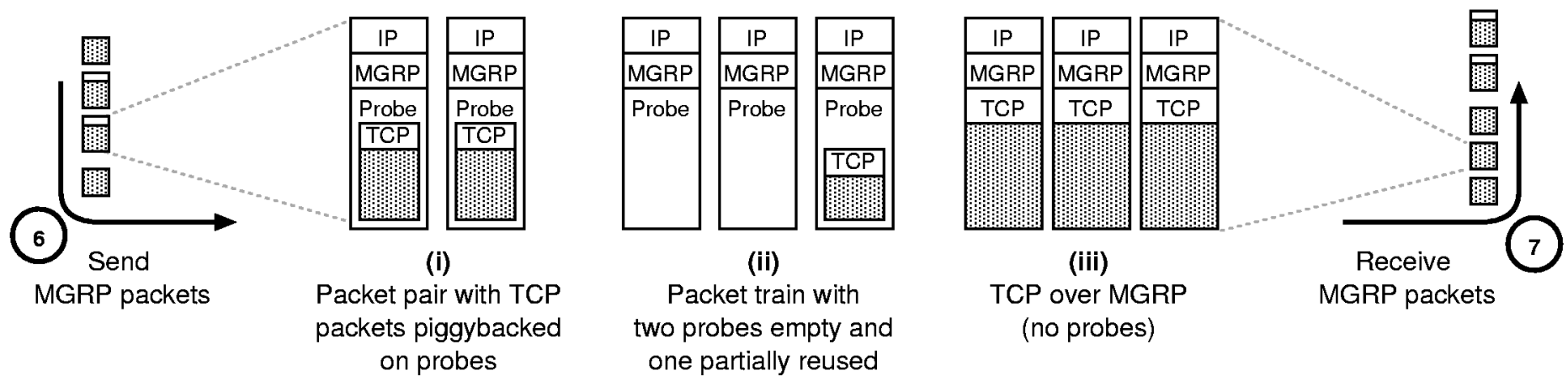
Architecture: Step-by-Step Example



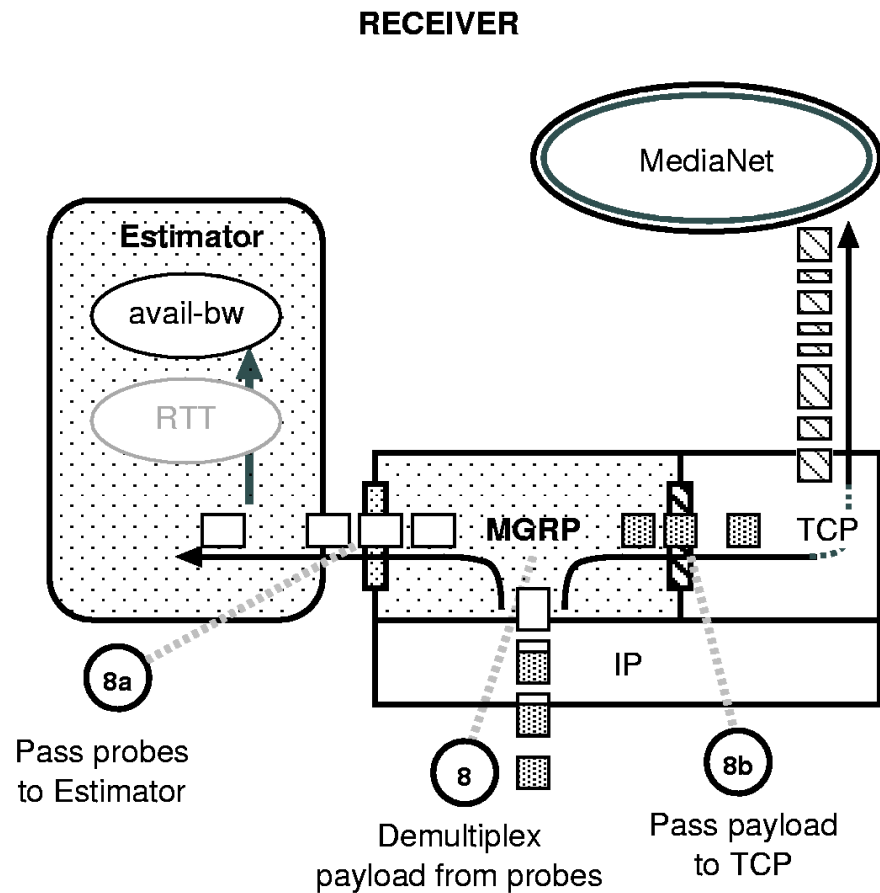
Architecture: Step-by-Step Example



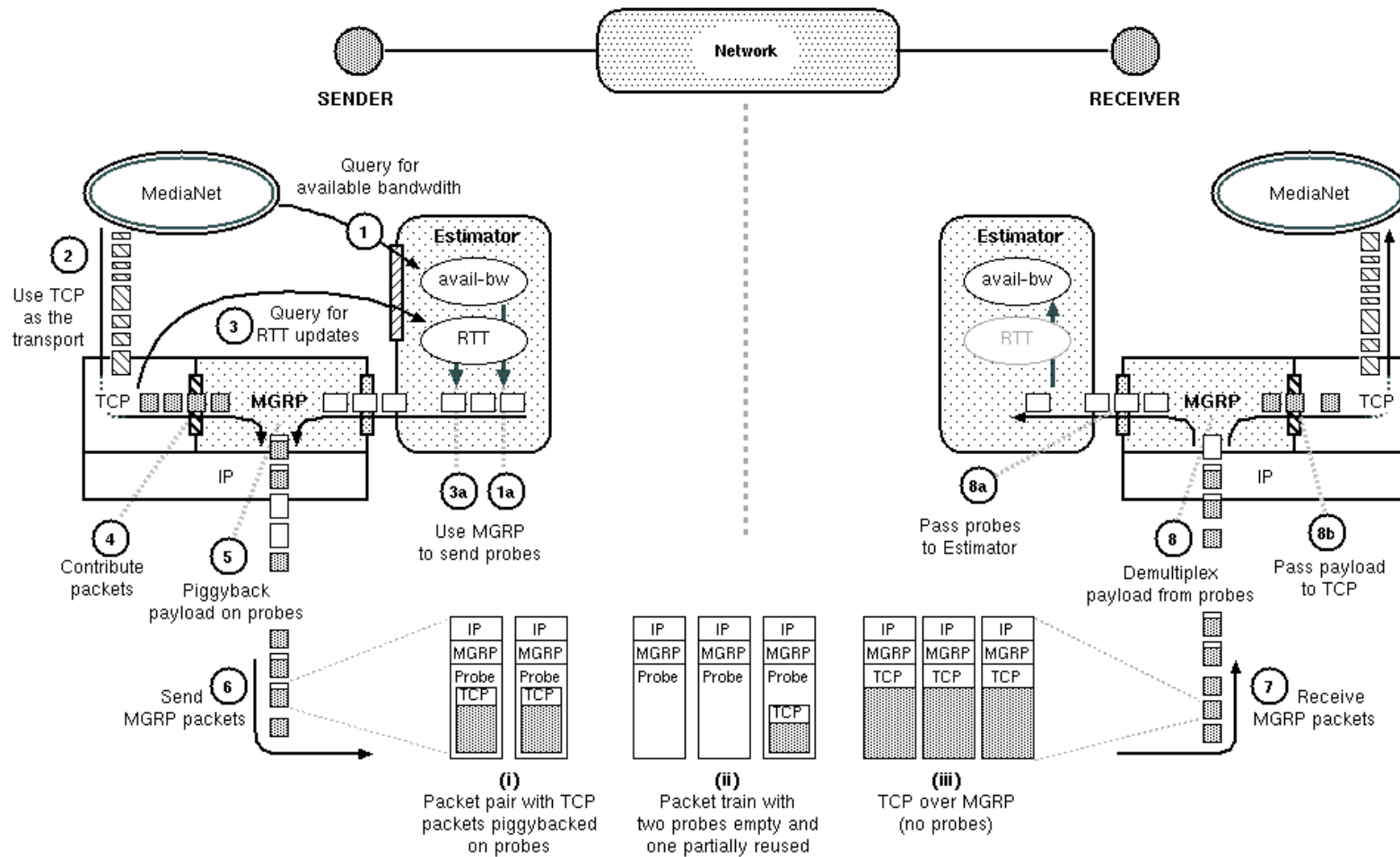
Architecture: Step-by-Step Example



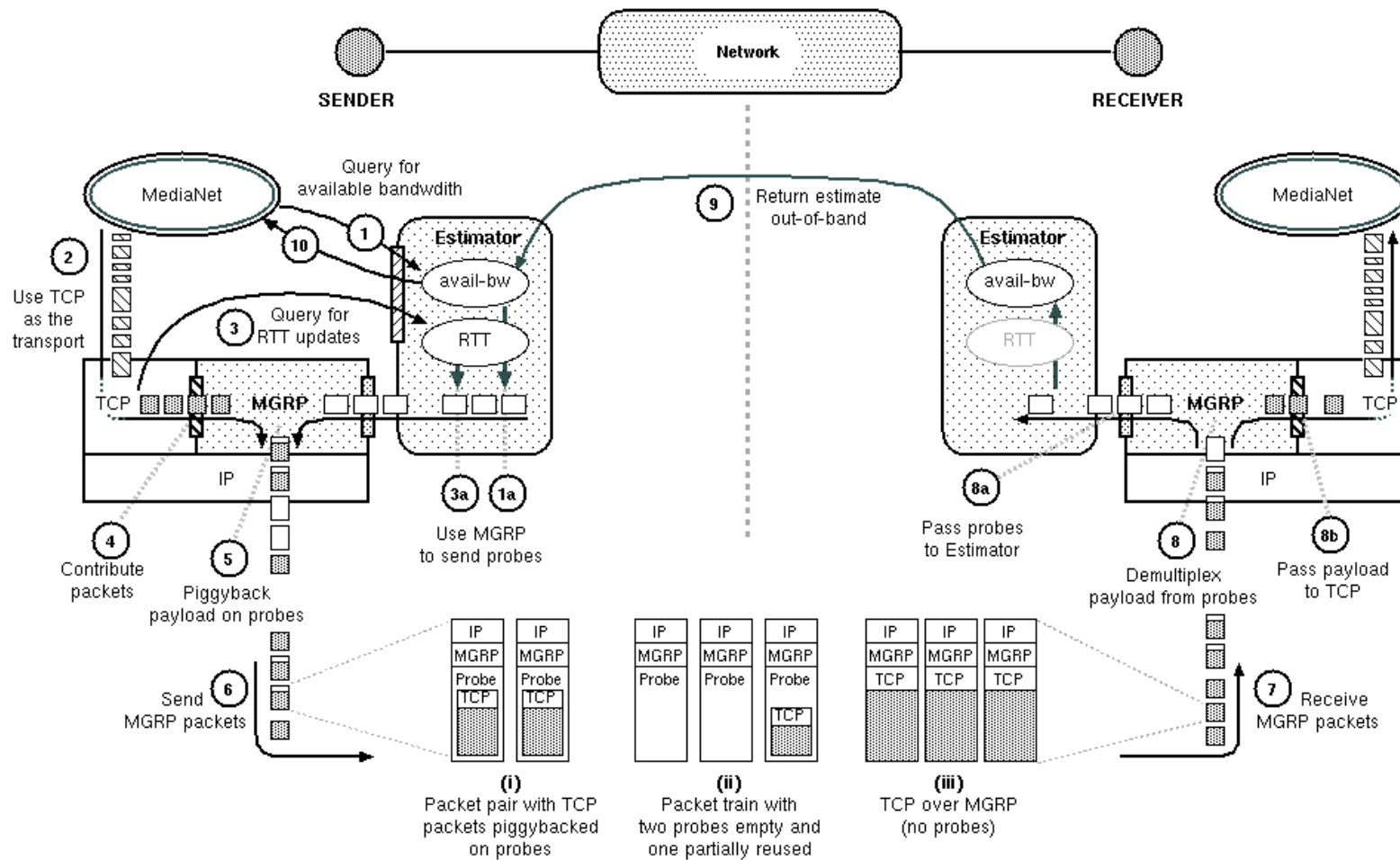
Architecture: Step-by-Step Example



Architecture: Step-by-Step Example



Architecture: Step-by-Step Example



Work Completed

- **Basic Design of Measurement Manager architecture**
 - Estimator component (estimation algorithms)
 - Probing component (probing primitives)
- **Implemented *Probing* component**
 - Designed New Layer-4 Protocol: MGRP
 - Probes can be piggybacked on transport payload
 - Programming interface that exports *probing primitives*
- **Prototype implementation for the *Estimator***
 - Adapted Pathload algorithm to piggyback on payload
- **Preliminary experiments with real traffic**
 - Measurement Accuracy
 - Impact on TCP
 - Probe Reuse Ratio

Proposed Work Overview

- **Evaluate the Measurement Manager Architecture**
 - *Efficiency*: How much bandwidth is saved?
 - *Modularity*: Can the estimation services be reused?
 - *Flexibility*: How flexible are algorithm designers?
- **Study trade-offs and optimizations**
 - Estimation algorithms
 - Transport Protocols
- **Implement the *Estimator***
 - Add more algorithms
 - Automate algorithm selection based on client requirements
 - Create programming interface for clients
- **Extend MGRP**
 - Enable one-way active probing

Proposal Objectives

Motivation

Why do we need the measurement manager?

Architecture

Step-by-step example to present the architecture

Current Implementation & Preliminary Results

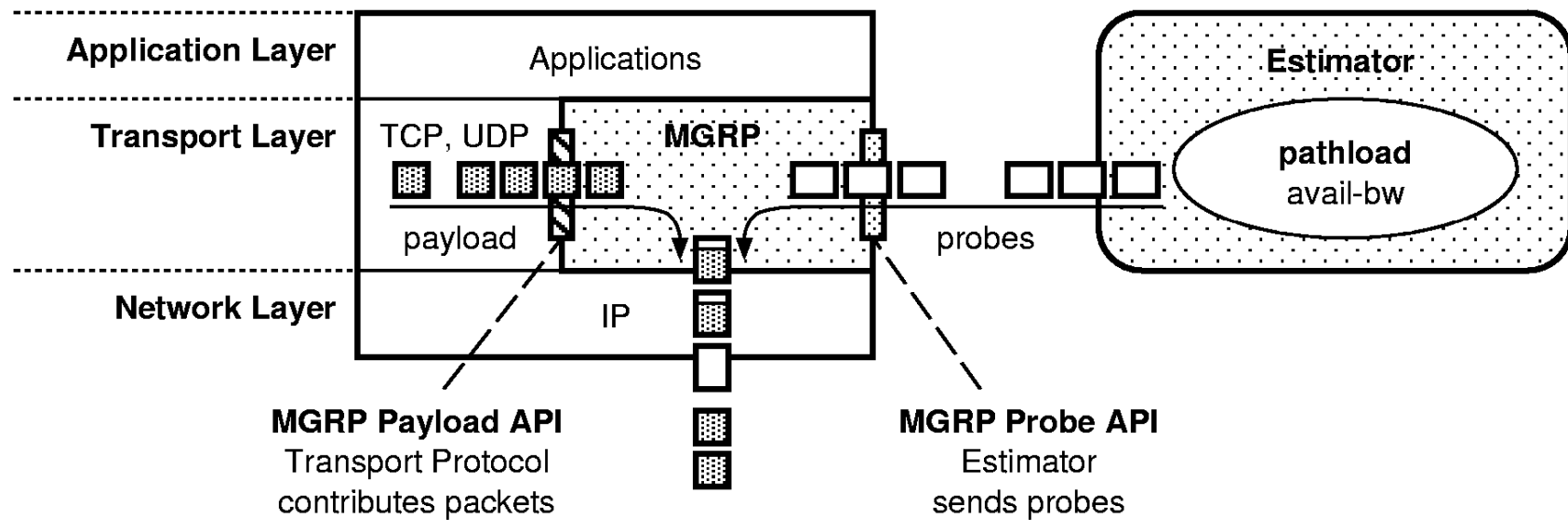
Proposed Work

Divided into three phases

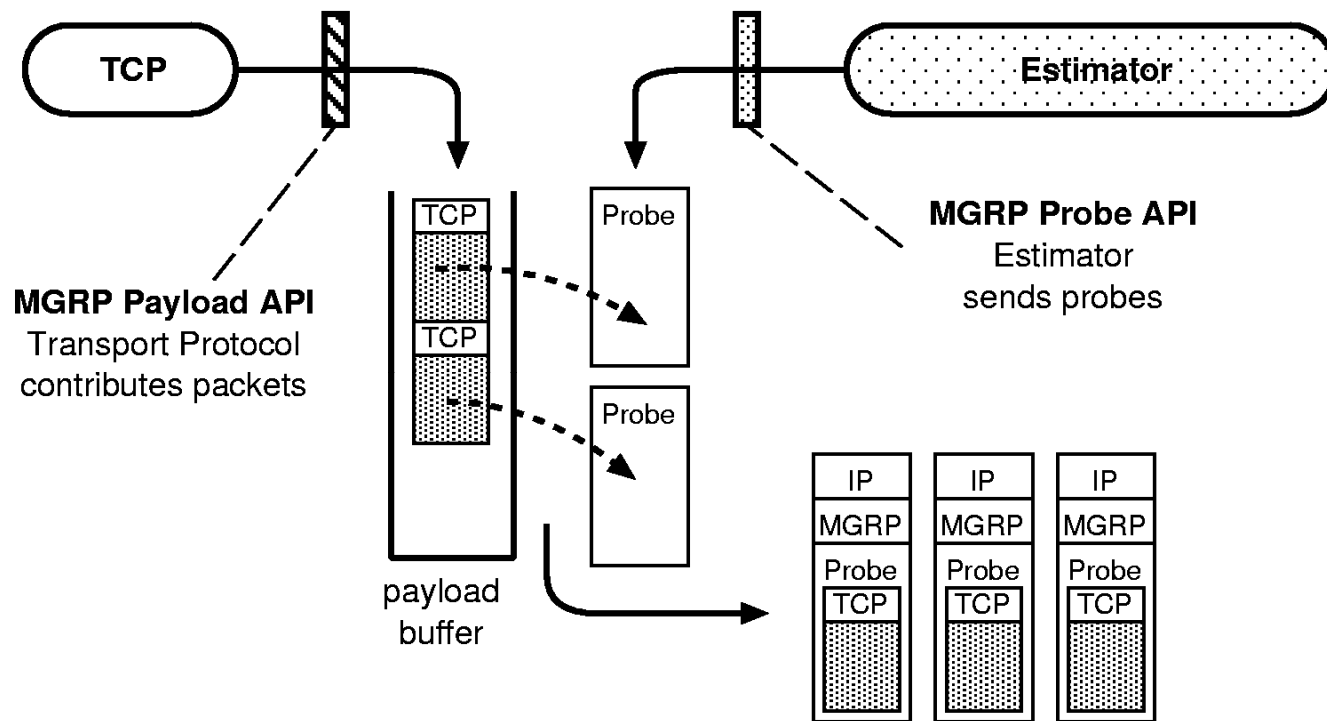
Current Implementation

- **MGRP protocol**
 - Piggybacks TCP payload on probes
 - Exports probing primitives
 - Layer-4 protocol with IP protocol number 254
 - Implemented in the Linux kernel
- **Estimator service**
 - Exports one measurement service
 - Measures available bandwidth (pathload algorithm)
 - Implemented in userspace

Current Implementation

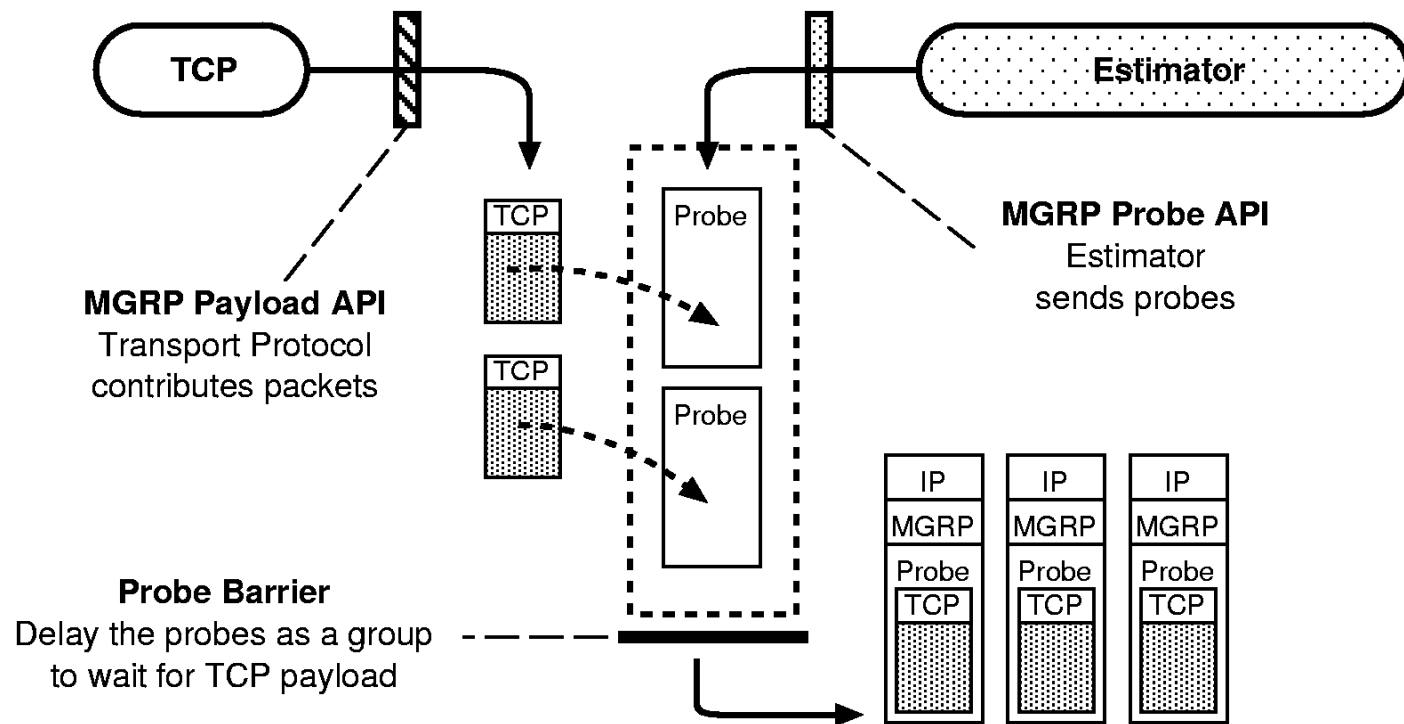


Piggybacking: Delay the payload



- TCP packets are buffered in a FIFO buffer
- Increases chances that probes will find payload for piggybacking
- Impact on TCP: appears as increased RTT

Piggybacking: Delay the probes



- Probes are delayed waiting for TCP payload
- No impact on TCP
- Algorithms can compensate for delay

Pathload

- Active measurement tool
 - By Jain & Dovrolis at Georgia Tech
 - Measures end-to-end available bandwidth
 - *Self Loading Periodic Streams* methodology (SLoPS)
 - One way delays of a periodic stream show increasing trend when the stream rate is higher than the available bandwidth
- Operates in rounds
 - Sends packet trains of 100 UDP probes
 - Each packet train corresponds to a probing rate
 - Uses equally sized packets and uniform packet gaps

Pathload: Modified Operation

- Good candidate for our evaluation
 - Available bandwidth is a very useful network property
 - Quite accurate (even for GigE speeds, PAM05)
 - Non-trivial overhead (we can test probe reuse)
 - 100 probes per train contain a lot of wasted padding
- Modified Operation
 - Uses the Probe API to schedule probes with MGRP
 - Probes are generated inside the kernel
 - Timestamps are fixed appropriately
 - MGRP may piggyback TCP payload on the probes
 - MGRP may delay the packet train to increase reuse

Preliminary Experiments

- Effects of Piggybacking
 - Pathload accuracy
 - TCP performance
- Benefits of piggybacking
 - Show how payload is piggybacked on probes
- Tradeoffs
 - Effect of *payload buffer* delay on probe reuse
 - Effect of MGRP overhead

Proposal Objectives

Motivation

Why do we need the measurement manager?

Architecture

Step-by-step example to present the architecture

Current Implementation & Preliminary Results

Proposed Work

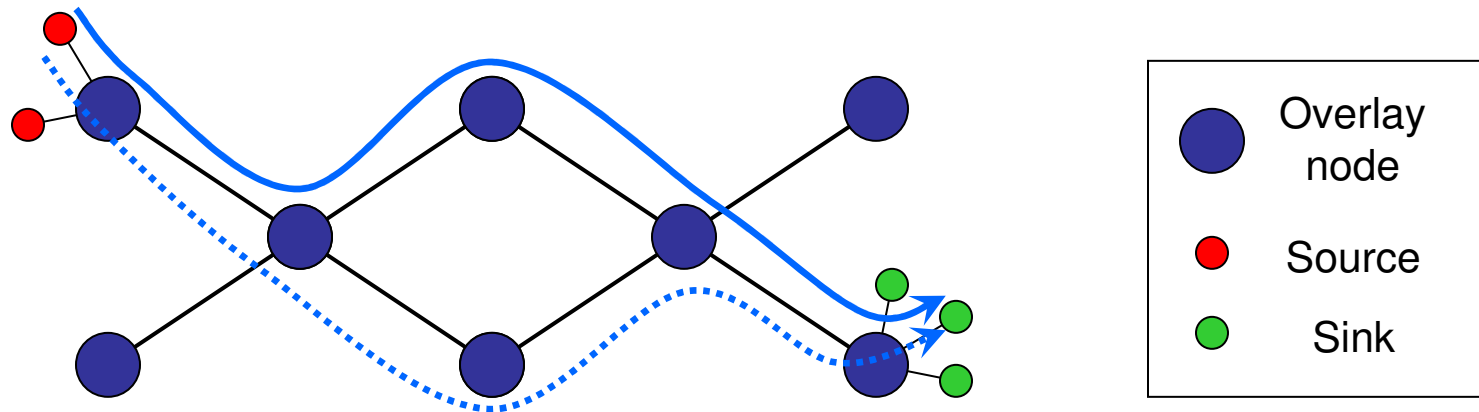
Divided into three phases

Proposed Work: Three Phases

- **Phase 1: Optimize Overlay Measurement**
 - Overlay construction and maintenance requires continuous measurement of paths between peers
 - Case Study: MediaNet
- **Phase 2: Optimize File Downloads**
 - Fast download times require intelligent selection of sources
 - Case Study: BitTorrent (peer-to-peer)
 - Case Study: Aria2 Download Manager (server selection problem)
- **Phase 3: Optimize TCP**
 - TCP not optimized for every network environment
 - Use measurement to characterize the network path
 - Switch to the best-suited congestion control algorithm
 - Example: differentiate between losses due to errors and due to congestion (as is the case in wireless environments)

Phase 1: Optimize Overlay Measurement

- Case Study: MediaNet



- Forwards media streams based on path conditions
 - Small number of long-term TCP connections between peers
- Periodic measurement of active paths
 - MGRP will use existing data to lower overhead
- Fast measurement of inactive paths
 - The Estimator will use faster algorithm with higher overhead
- Measurement manager picks suitable algorithm

Phase 1: Goals

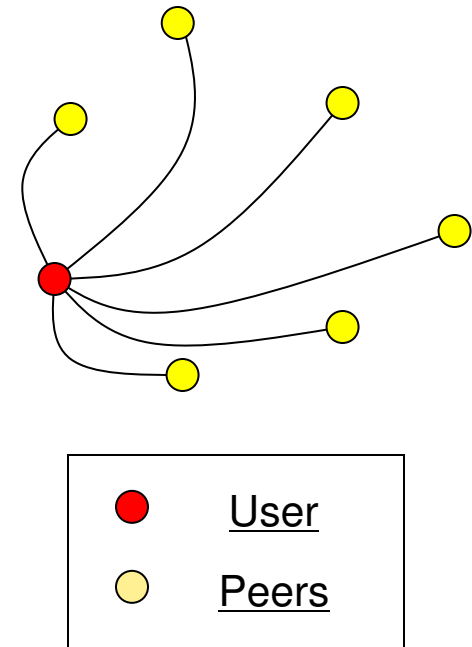
- **Implement the *Estimator***
 - MediaNet nodes query local *Estimator*
 - Measurement is provided as a service
 - Add capacity estimation
 - Create the Estimator Programming Interface
- **Evaluate the benefits of the Measurement Manager**
 - *Modularity*: Any other overlay can use the *Estimator*
 - *Efficiency*: Measurement with minimal overhead for paths with existing overlay traffic
 - As the *Estimator* service improves (better algorithms) so does the overlay (it performs better without any change)
- **Trade-offs**
 - Increase probe reuse by delaying TCP payload
 - Increase probe reuse by delaying probes

Phase 1: Timetable

Optimizing overlay measurement (in progress: 6 weeks remaining)		
Design	Architecture	completed
Implementation	MGRP Protocol	completed
	Available bandwidth (pathload)	completed
	Delay probes instead of payload	completed
	Capacity (pathrate)	almost complete
	Integrate MediaNet	1 week
	Estimator service	1 week
Evaluation	Experiments	3 weeks
Paper writing		1 week

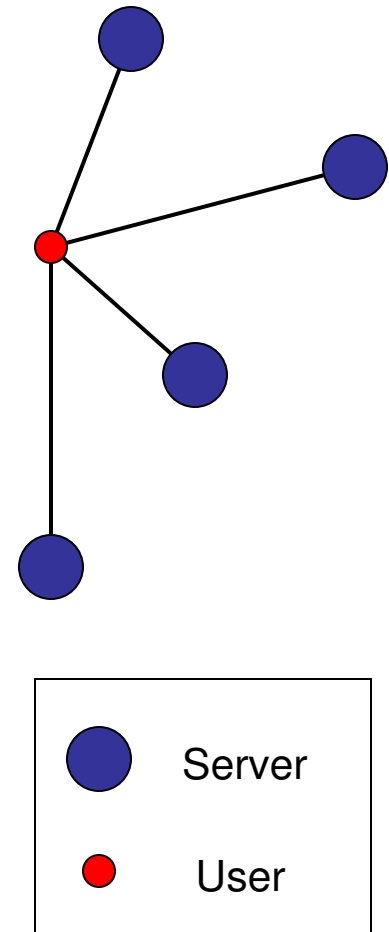
Phase 2: Optimize File Downloads

- Case Study: BitTorrent
 - **Original operation**
 - Uses segmented downloading and uploading between multiple peers to shorten the total download time
 - Peer selection at random with minimal measurement
 - Multiple short TCP connections
 - **Modified operation**
 - Find best download and upload peers by measuring while exchanging segments
 - Detect shared bottlenecks and do not use peers that are behind the same bottleneck
 - Use fast algorithms to prune the worst peers and then select among the rest



Phase 2: Optimize File Downloads

- Case Study: Download Managers
 - **Original operation**
 - Uses segmented downloading from multiple mirror locations (server selection problem)
 - Server selection based on TCP throughput
 - Multiple medium-term TCP connections
 - **Modified operation**
 - Use Estimator to evaluate paths based on multiple network properties: capacity, latency, loss, shared congestion
 - Measurement phase picks a subset of good servers followed by download phase
 - During measurement phase multiple servers are probed while all along downloading segments
 - Download phase keeps monitoring the path and changes if conditions deteriorate



Phase 2: Goals

– **Extend the *Estimator***

- Algorithm auto-selection based on user constraints
- Add lower-overhead available bandwidth algorithm (pathchirp)
- Add bottleneck detection algorithm (pathneck)
- Create a quality measure to characterize estimation algorithms

– **Extend MGRP**

- Implement one-way active probing
- Each end host can implement the *Estimator* independently (necessary for client-server download)

– **Evaluate the Measurement Manager**

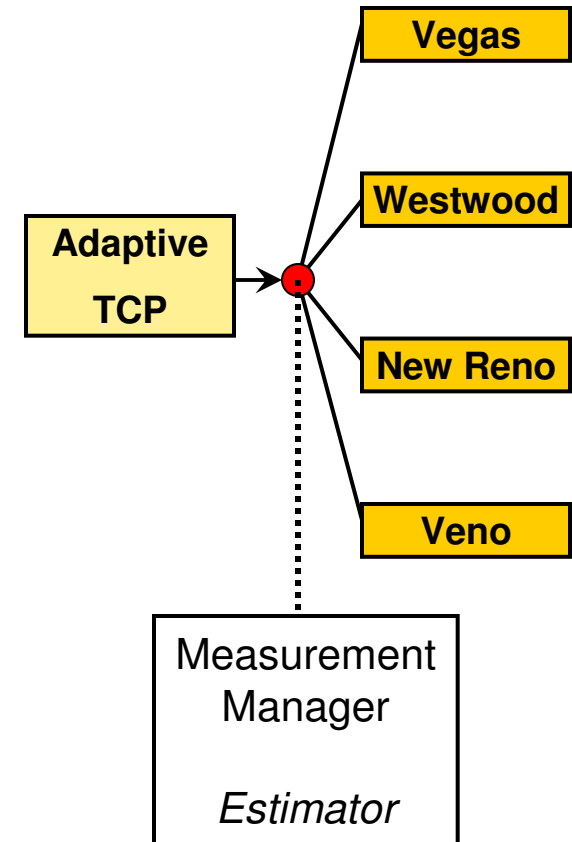
- *Efficiency*: Measure the path with your own download traffic (show how BitTorrent and downloading finishes faster)
- *Flexibility*: Algorithm design straightforward (pretend to be active)
- *Modularity*: Same measurement tools used by different application

Phase 2: Timetable

Optimizing file downloads (23 weeks)		
Estimator API	Quality Measure	2 weeks
	Algorithm auto-selection	3 weeks
Implementation	One-way active probing	3 weeks
	Integrated download manager	2 weeks
	Integrate BitTorrent	3 weeks
	Detect bottlenecks (pathneck)	2 weeks
	TCP (that sends only on probes)	1 week
Evaluation	Experiments	4 weeks
Paper writing		3 weeks

Phase 3: Optimize TCP

- Case Study: Adaptive TCP
 - **Original operation**
 - Users must select TCP congestion algorithm manually
 - **Modified operation (Adaptive TCP)**
 - Start using a default congestion algorithm
 - Use Estimator to discover the properties of the network path as you send data
 - Switch to the congestion control algorithm best suited for the network conditions
 - Example: If you detect a wireless segment in your path, switch to an algorithm designed for wireless
 - **Delegate measurements to *Estimator***
 - RTT estimates
 - Loss detection (due to error or congestion?)



Phase 3: Goals

- **Create an Adaptive TCP algorithm**
 - Auto selects among specialized congestion control algorithms
- **Evaluate the Measurement Manger**
 - Modularity: Even transport protocols can use the *Estimator* (estimation algorithms can be reused too)
- **Extend the *Estimator***
 - Add RTT estimation
 - Add Loss detection and differentiation
- **Study Impact of MGRP on TCP**
 - Piggybacking can be optimized with:
 - TCP packet buffering
 - TCP packet segmentation
 - Micro-gaps between TCP packets
 - Relate with TCP Pacing work

Phase 3: Timetable

Optimizing TCP (22 weeks)		
Design	Adaptive TCP	3 weeks
Implementation	Replace RTT estimation	2 weeks
	Replace loss detection	2 weeks
	Integrate TCP Vegas (delay)	2 weeks
	Integrate TCP Westwood (bandwidth)	2 weeks
	Integrate TCP Veno (packet loss)	2 weeks
Evaluation	Experiments	6 weeks
Paper writing		3 weeks

Related Work:

Measurement Services

- Perform Measurements
 - **Periscope**: programmable probing framework
 - **Scriptroute**: conduct measurement from multiple vantage points
 - **Pktd**: packet capture and injection agent
 - **NIMI**: large-scale measurement infrastructure
- Query for Measurements
 - **iPlane**: overlay service that predicts path properties
 - **Sophia**: information plane for networked systems
 - **Routing Underlay**: shared measurement infrastructure
- **Congestion Manager**
 - Inspiration for Measurement Manager

Related Work: Inline Measurements

- Reuse TCP packets as probes
 - **TCP Probe:** uses back-to-back TCP packets as probes (integrates TCP with CapProbe algorithm)
 - **ImTCP:** spaces TCP packets and uses them as packet trains (to estimate bandwidth algorithm)
- Inject probes in a TCP connection
 - **TCP Sidecar:** masks probes as TCP retransmissions within a TCP stream
 - **Sting:** manipulates a TCP connections to measure packet loss rates
 - **Sprobe:** manipulates a TCP connection to send packet pairs to estimate bottleneck bandwidth

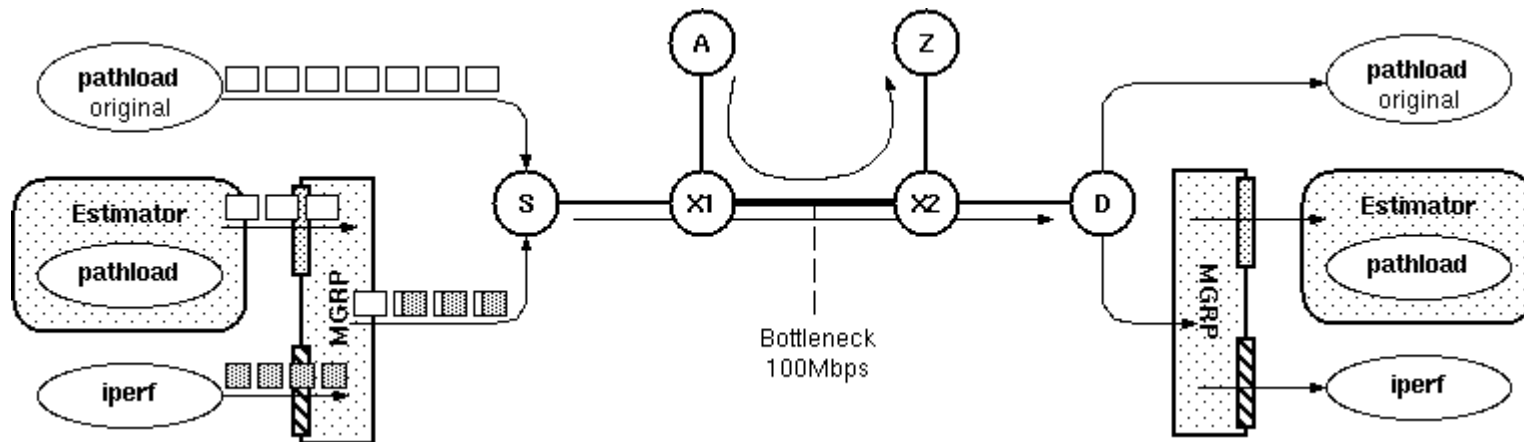
How we stand out

- **Efficiency and Accuracy**
 - Balances the **accuracy of active measurement** and the **efficiency of passive measurement**
- **Flexibility**
 - Overlays, applications and transport protocols can use the same set of measurement services
- **Modularity**
 - Estimation algorithms can be designed as if active, but transparently implemented as if passive, independent of transport protocols and applications
- **Deployment**
 - Probing and Estimation modules can deploy and evolve independently

Summary

- **Network measurement is inefficient and inflexible**
 - All network applications need to measure the network
 - Current practices do not scale
- **Proposed the Measurement Manager**
 - Provides measurement as a service
 - Separates estimation from probing
 - Uses transport payload to reduce probing overhead
 - Independent of applications and transport protocols
- **Proposed work in three phases**
 - Optimize Overlay Measurement
 - Optimize File Downloads
 - Optimize TCP

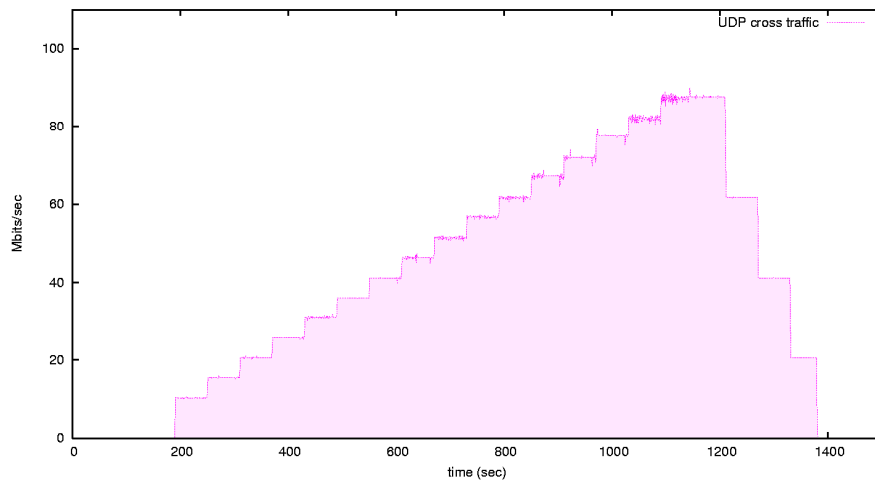
Preliminary Experiments: Topology



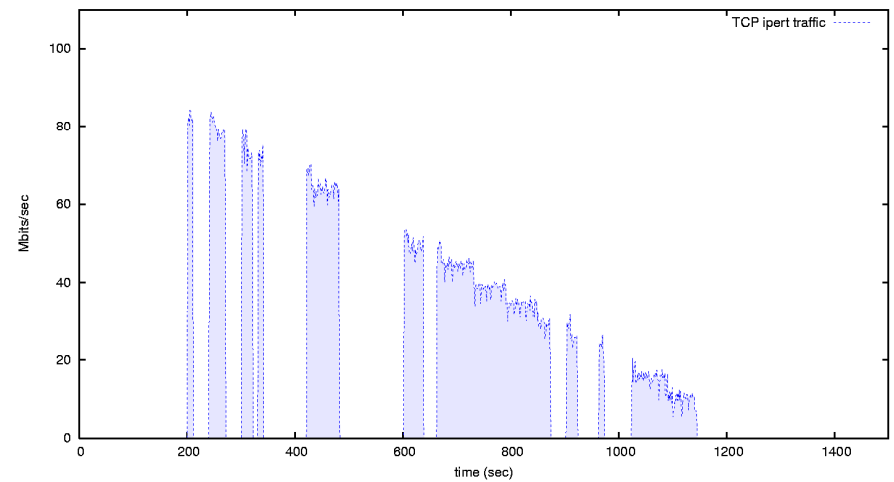
- UDP cross traffic A to Z
- TCP iperf traffic from S to D
- Original pathload (over UDP) from S to D
- Modified pathload (over MGRP) from S to D

Instantaneous traffic at Bottleneck

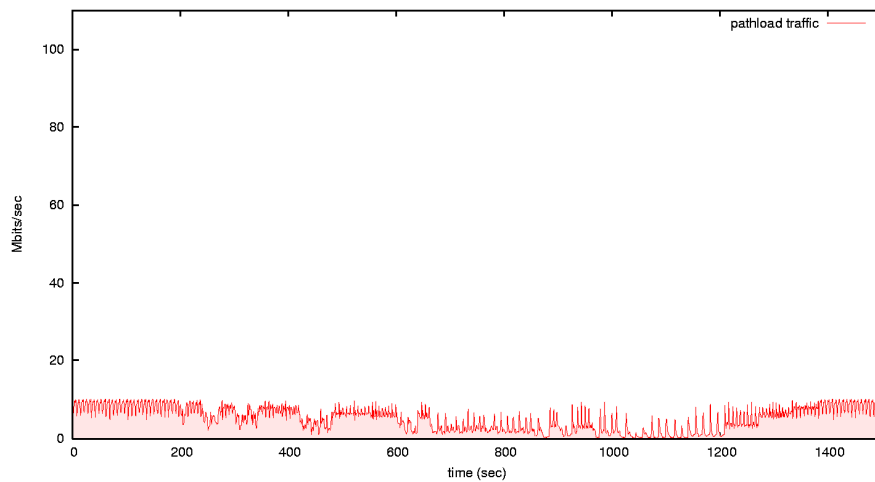
(a) UDP cross traffic



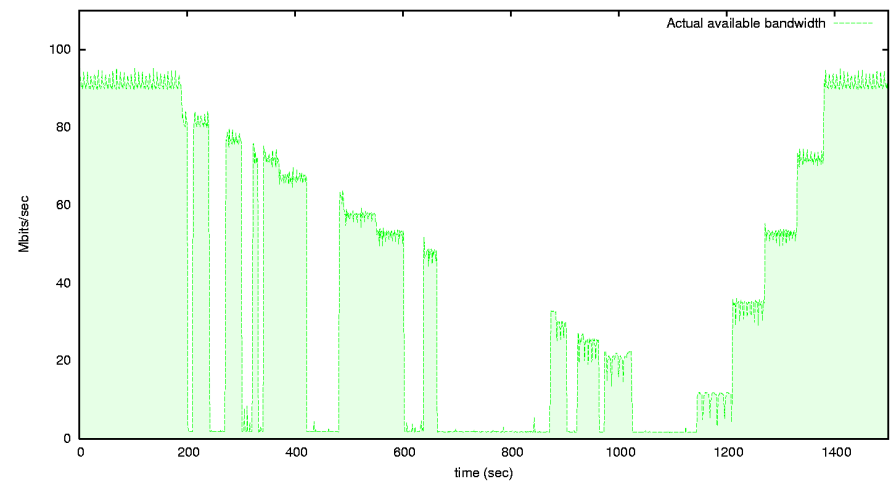
(b) TCP iperf application traffic



(c) Pathload probe traffic

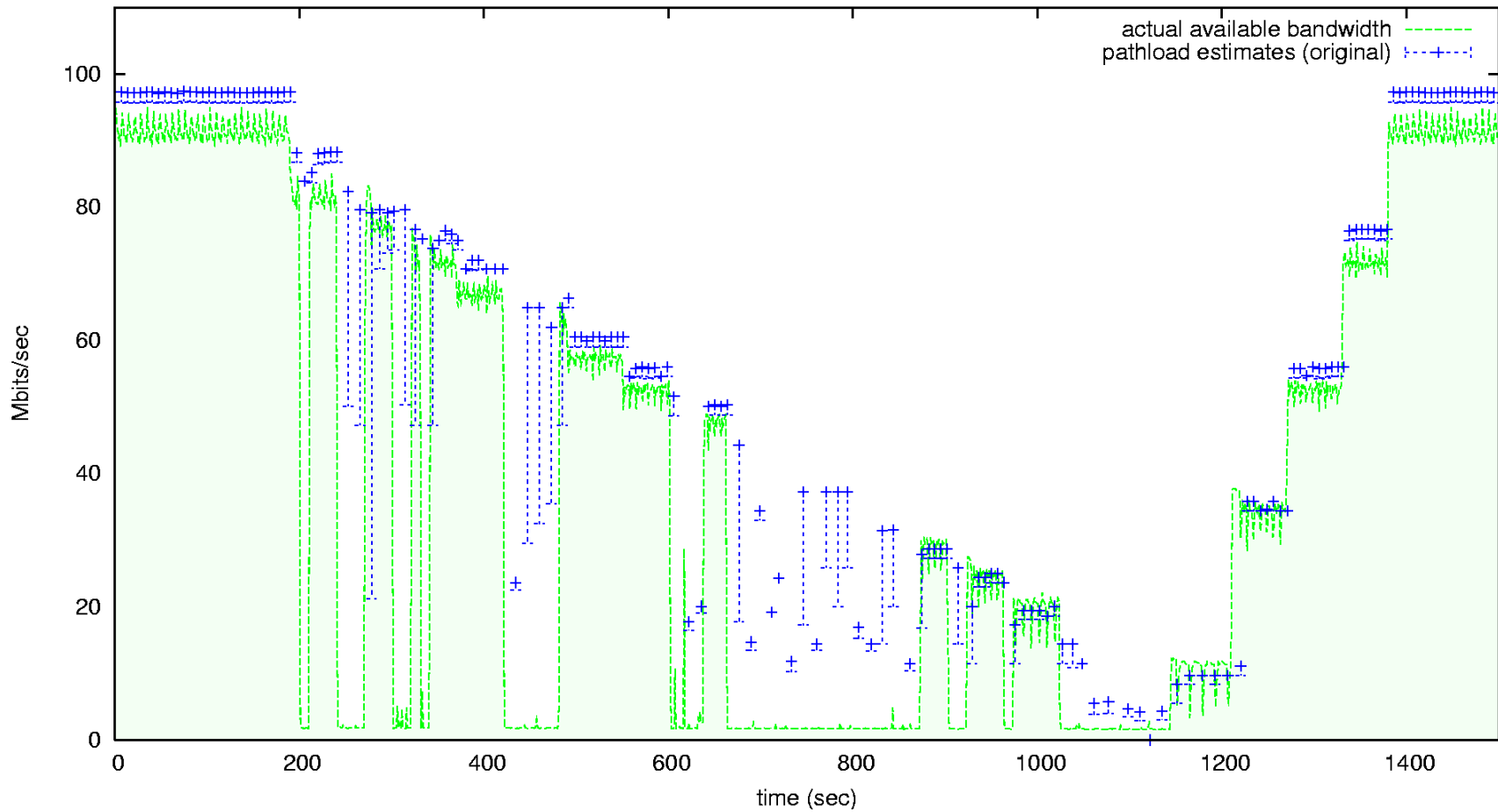


(d) Actual available bandwidth



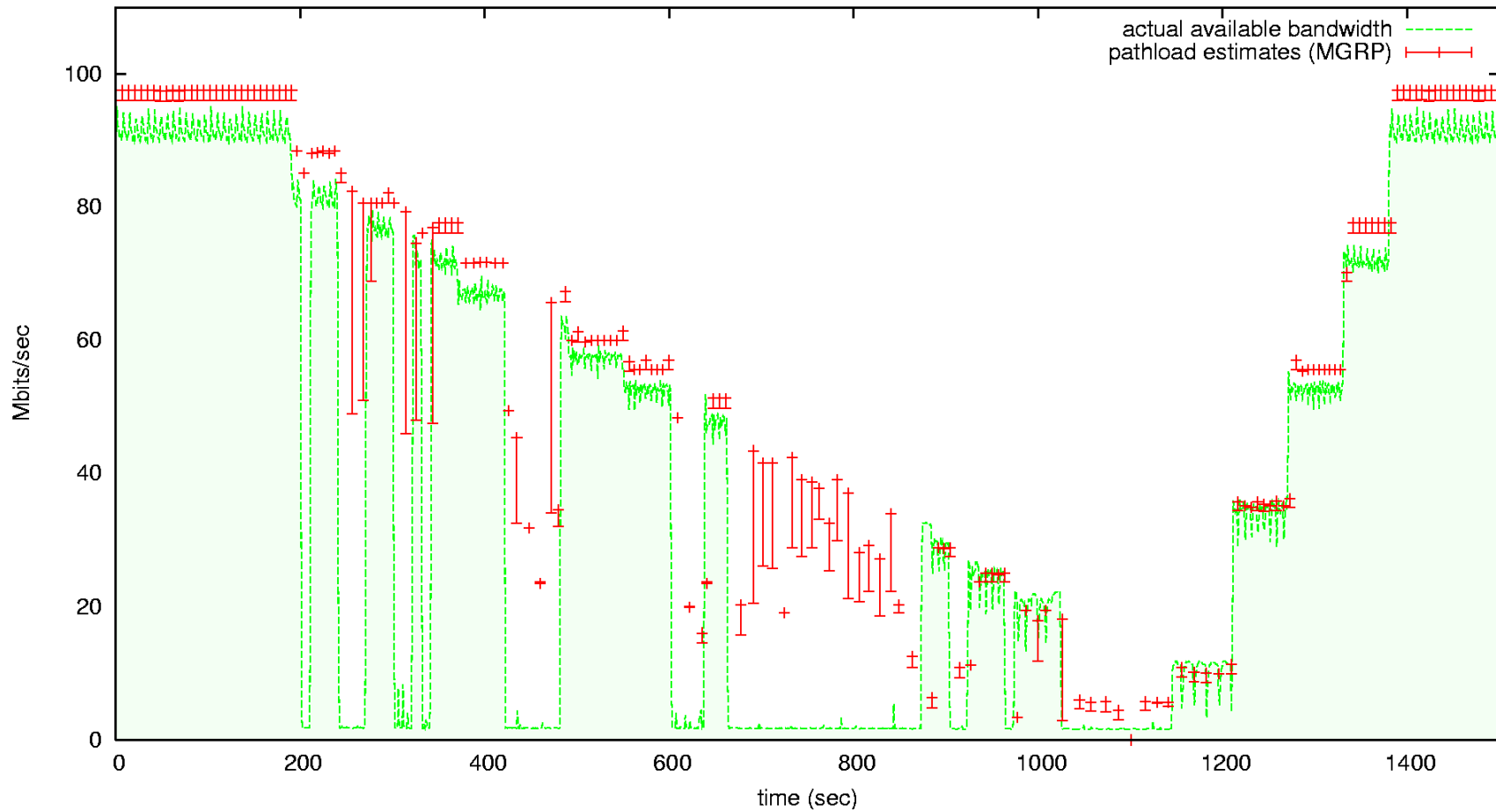
Pathload accuracy

Original over UDP



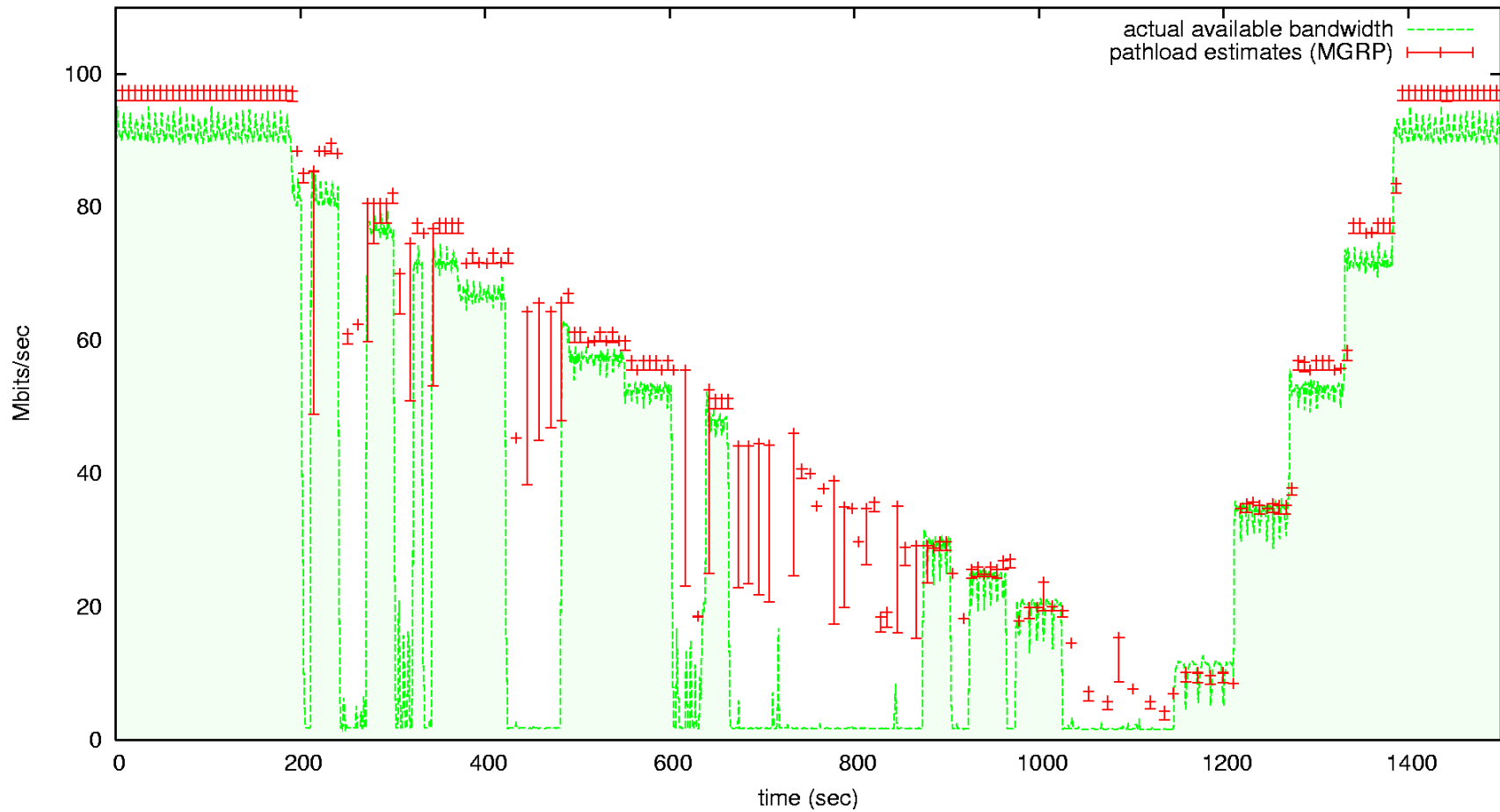
Pathload accuracy

Modified over MGRP with no piggybacking



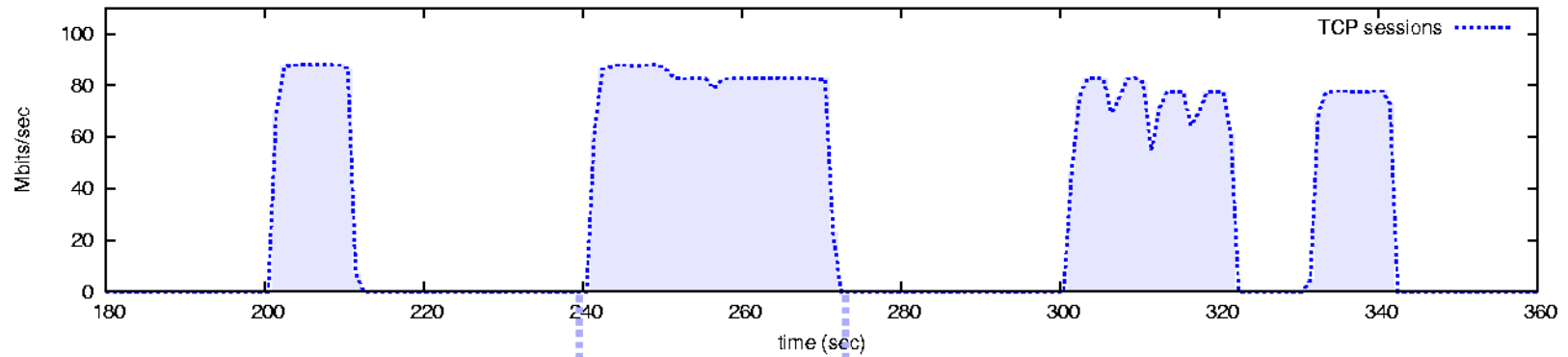
Pathload accuracy

Modified over MGRP with maximal piggybacking

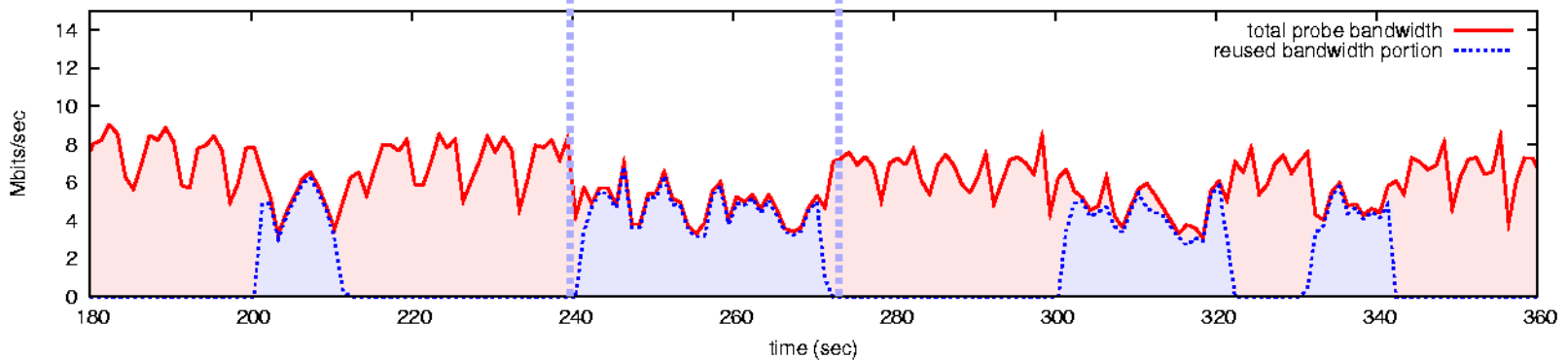


Piggybacking: full reuse

(a) TCP sessions that contribute payload to MGRP

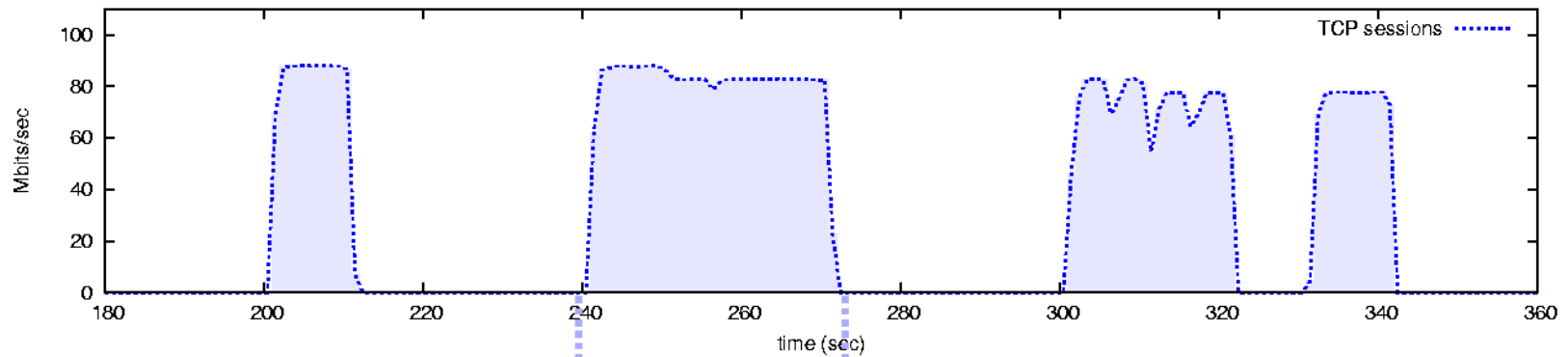


(b) 8ms buffer delay results in full probe reuse

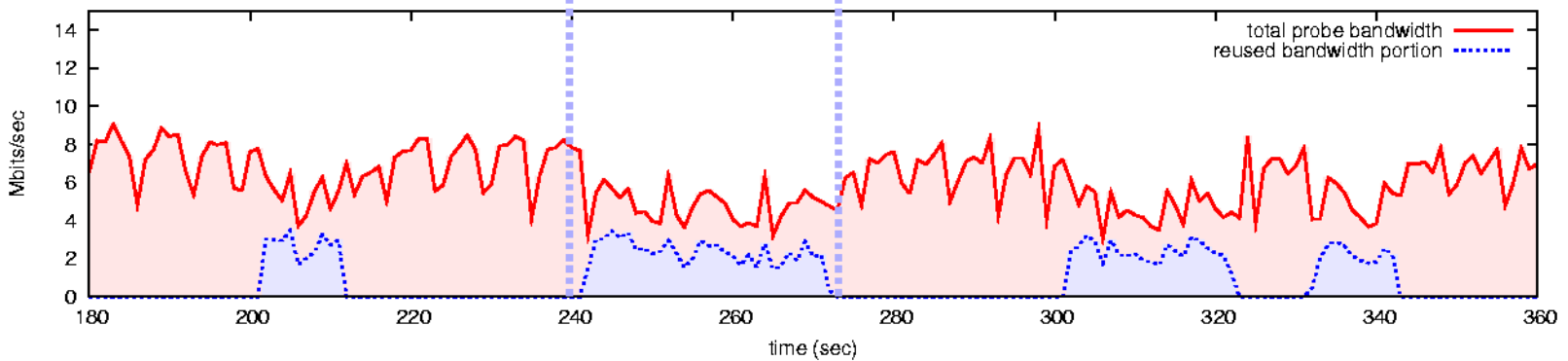


Piggybacking: partial reuse

(a) TCP sessions that contribute payload to MGRP

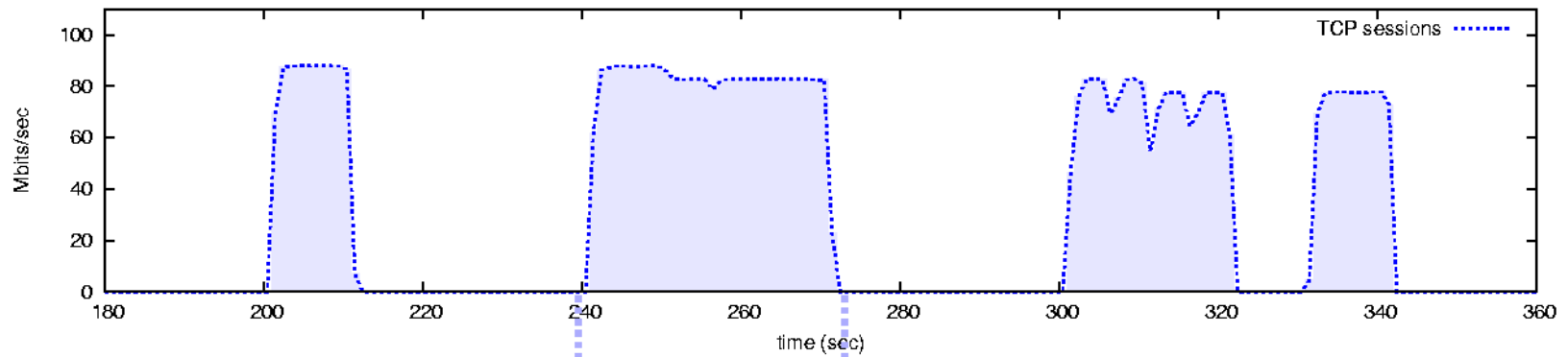


(c) 4ms buffer delay results in partial probe reuse

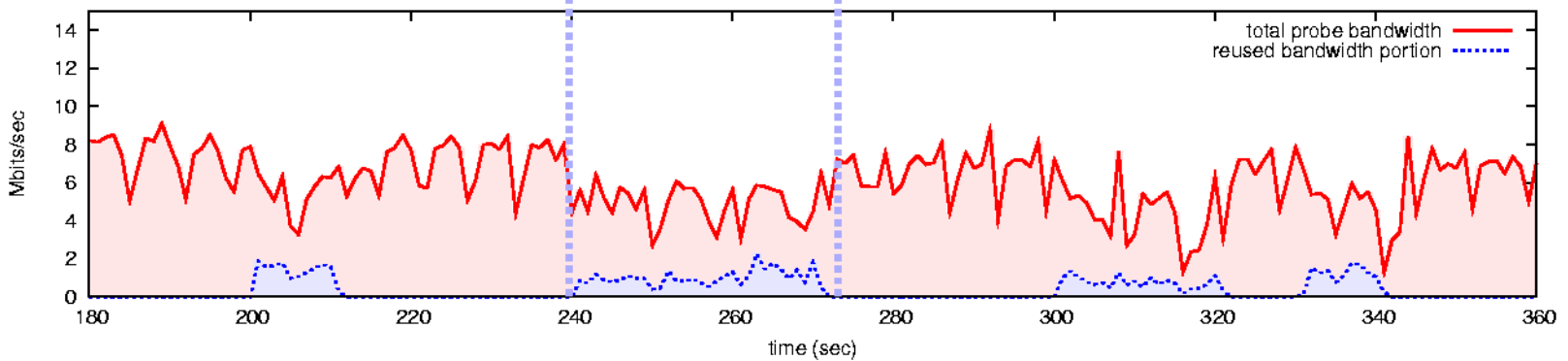


Piggybacking: minimal reuse

(a) TCP sessions that contribute payload to MGRP

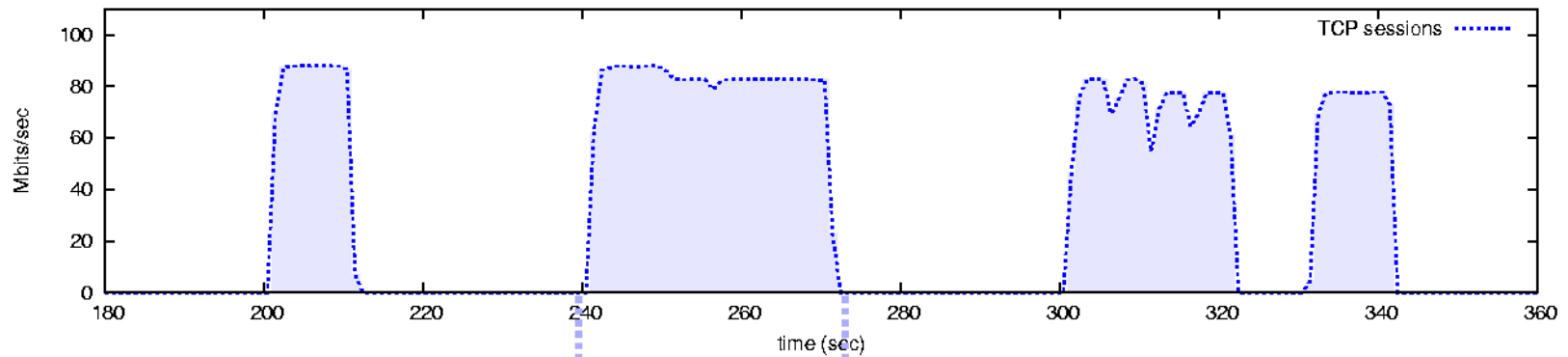


(d) 2ms buffer delay results in minimal probe reuse

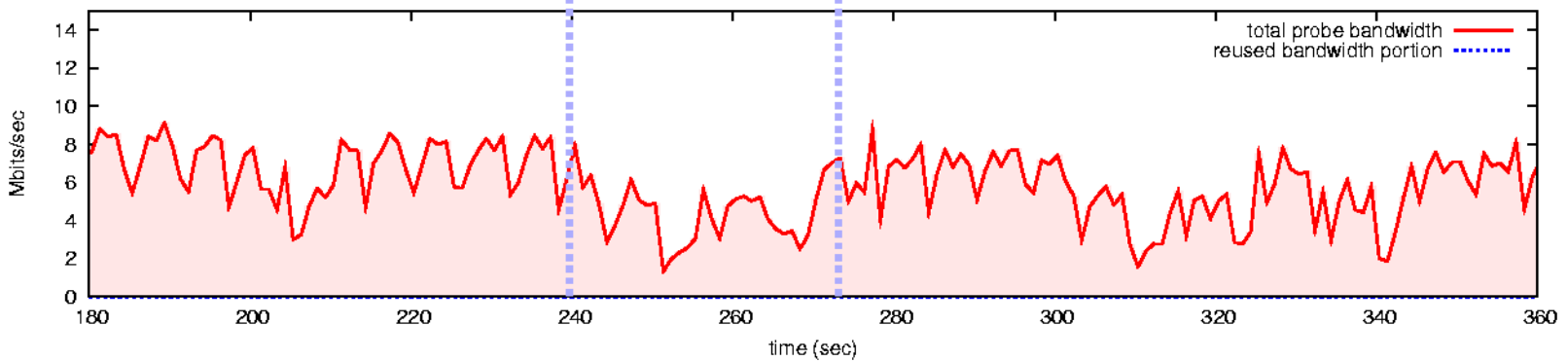


Piggybacking: no probe reuse

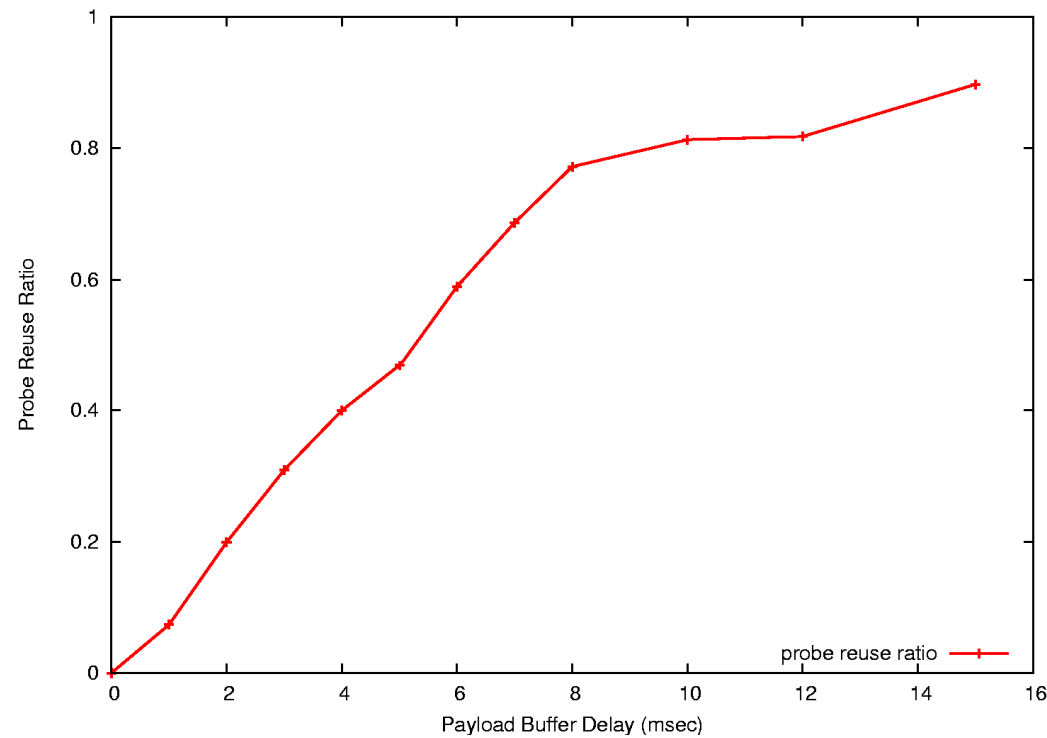
(a) TCP sessions that contribute payload to MGRP



(e) no buffer delay results in no probe reuse



Piggybacking: reuse ratio



The more payload is buffered
the more probes are reused

Sending Packet Trains

```
/* send a packet train using an MGRP probe transaction */

/* 10 packets of size 1000bytes each with 1ms gap */
int probe_size = 1000;
int probe_data = 20;
int probe_gap = 1000
int probe_pad = probe_size - probe_data;

char probe[probe_size];

/* pass information using ancillary data */
struct msghdr msg = {...};
struct mgrp_probe *probe = (pointer to msg_control buffer);

int sock = socket(AF_INET, SOCK_DGRAM, IPPROTO_MGRP);

/* pass first probe to MGRP: raise barrier */
probe->barrier = 1;
probe->pad = probe_pad;
probe->gap = 0
sendmsg(sock, &msg, 0);

/* pass probes 1..9 */
for (i = 1; i < 9) {
    probe->barrier = 1;
    probe->pad = probe_pad;
    probe->gap = probe_gap;
    sendmsg(sock, &msg, 0)
}

/* pass last probe: lower the barrier */
probe->barrier = 0;
probe->pad = probe_pad;
probe->gap = probe_gap;
sendmsg(sock, &msg, 0);

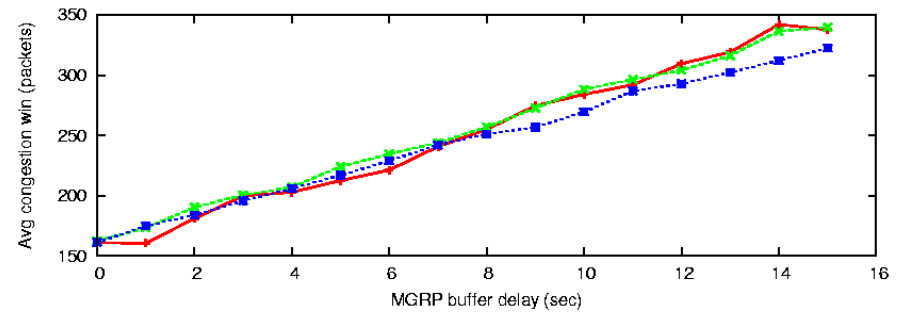
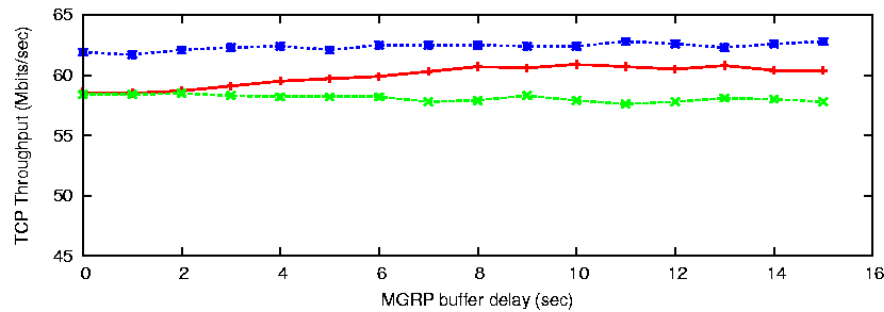
/* MGRP sends the packet train */
```

Impact on TCP

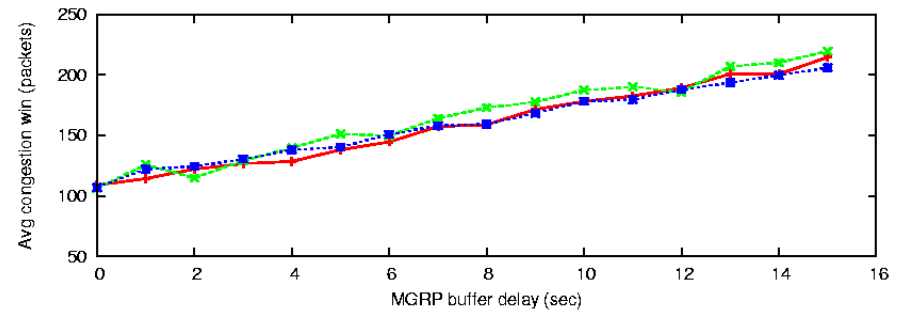
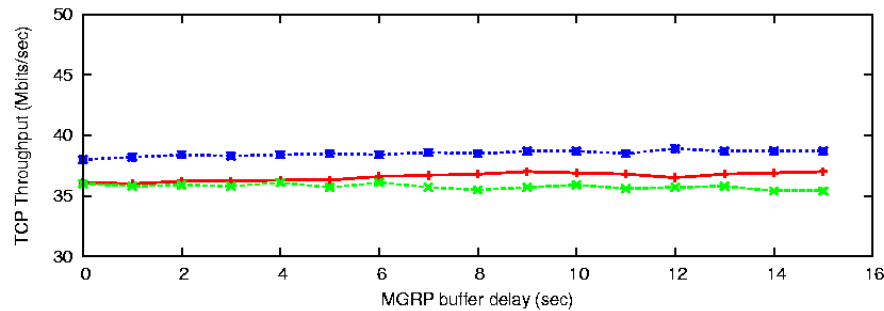
TCP Throughput

Average TCP
Congestion window

(a) TCP iperf session 7, 60sec



(b) TCP iperf session 17, 180sec



TCP original (emulate buffer delay with increased link delay) - - - + - - -
 TCP over MGRP with piggybacking - - - x - - -
 TCP over MGRP without piggybacking - - - * - - -