

FLEXible Middleware And Transports (FLEXMAT) for Real-time Event Stream Processing (RT-ESP) Applications

Joe Hoffert, Doug Schmidt
Vanderbilt University

Focus: Scalability, Flexibility

• Challenges and Goals

– Real-time Event Stream Processing (RT-ESP)

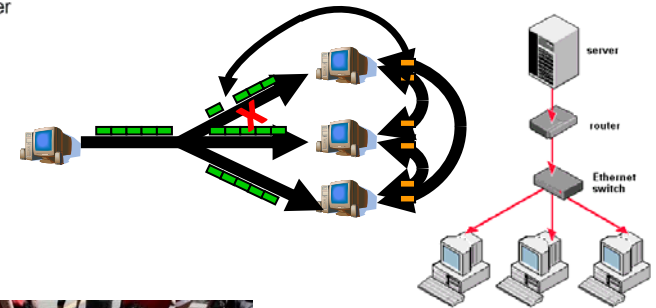
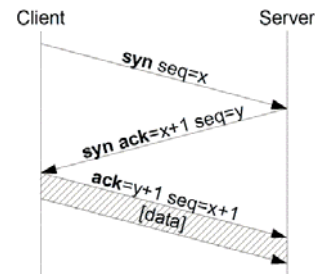
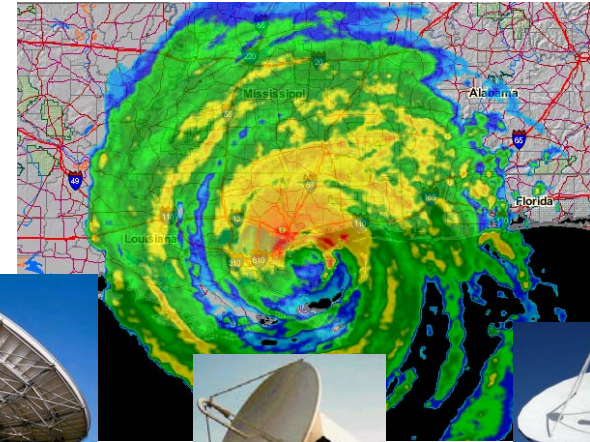
- Support mission critical systems
- Coordinate multiple event streams
 - Surveillance cameras
 - Temperature probes
 - Radars
 - Online stock trade feeds

– RT event notification middleware

- Promising platform
- Requires scalability, flexibility
- No single transport protocol is “silver bullet”

– Research needed

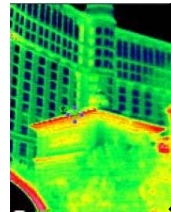
- Existing & new protocols
- Determine trade-offs, configuration, tuning



Motivating Example: Search and Rescue Mission

• Scenario

- Regional hurricane and consequent flooding
- Survivors trapped
- Search and rescue missions initiated
- Search application fuses multiple sensor streams
 - Thermal scans from unmanned aerial vehicle (UAV)
 - Video from existing camera infrastructure
 - Data streams sent to datacenter for fusion



Motivating Example: Search and Rescue Mission (cont.)

• Challenges & goals

- Dynamic ad-hoc environment
 - Many other missions and applications
 - Competition for scarce resources
 - Fluctuating resource availability

- Maintain specified QoS
 - As resources decrease, increase
 - Manage “antagonistic” QoS
 - Reliability vs. low latency
 - Soft real-time

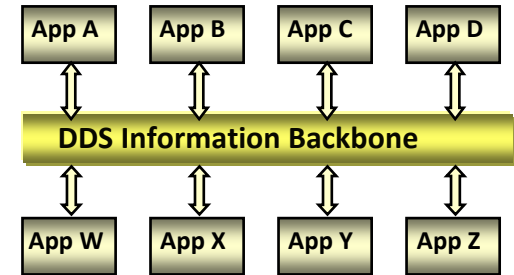
- Need systematic evaluation of RT event notification middleware and transport protocol



Solution Approach: FLEXible Middleware And Transports (FLEXMAT)

Data Distribution Service

- Robust standardized pub/sub API
- Fine-grained QoS policy control



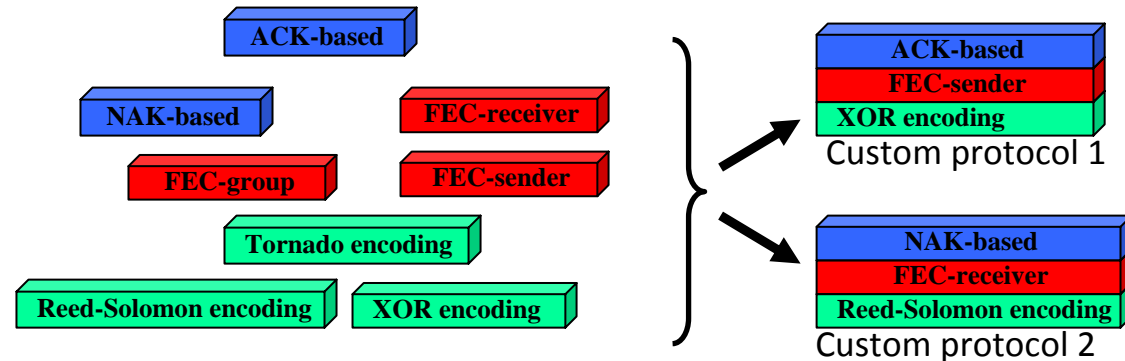
Targeting OpenDDS, OpenSplice Implementations

- Pluggable transport protocol frameworks
- Open source



Adaptive Network Transports (ANT) framework

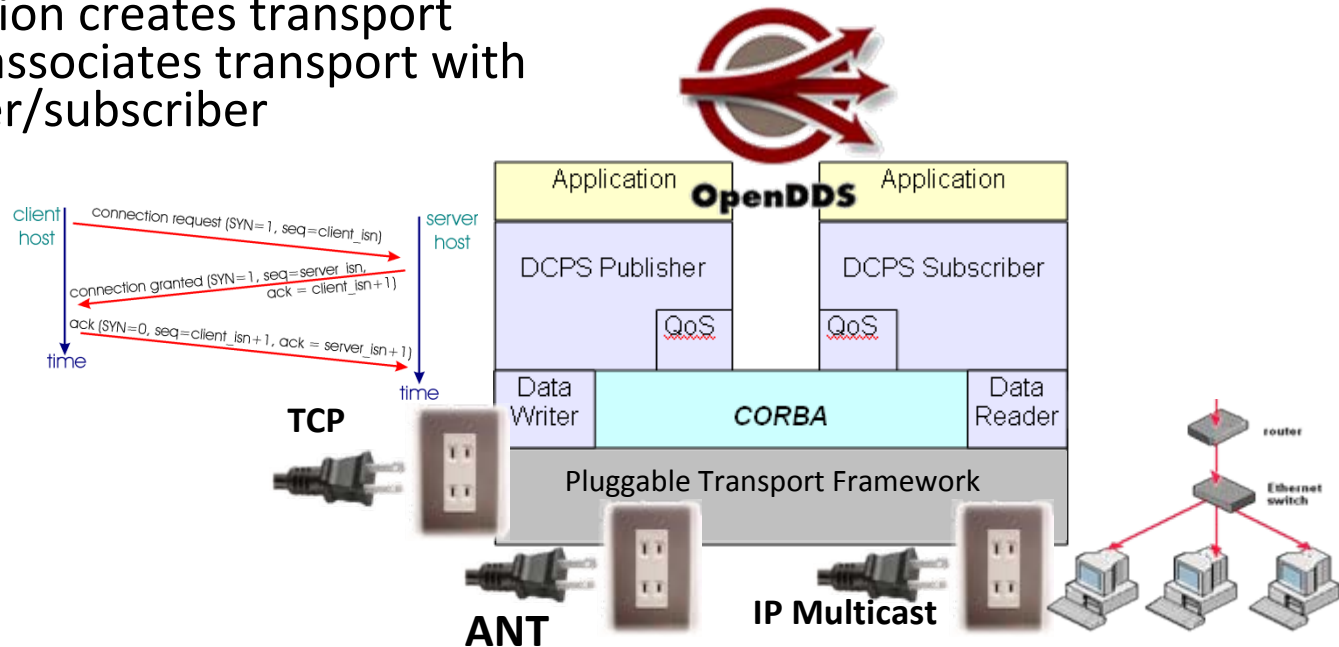
- Transport protocol framework
- Composable modules
- Fine-grained protocol control



OpenDDS Pluggable Transport Framework

OpenDDS Pluggable Transport Framework supports:

- Standard transport protocols (e.g., TCP, UDP, IP multicast)
- Reliable multicast protocol
- Custom transport protocols
 - Inherit from key classes
 - Define custom behavior
 - Application creates transport object, associates transport with publisher/subscriber

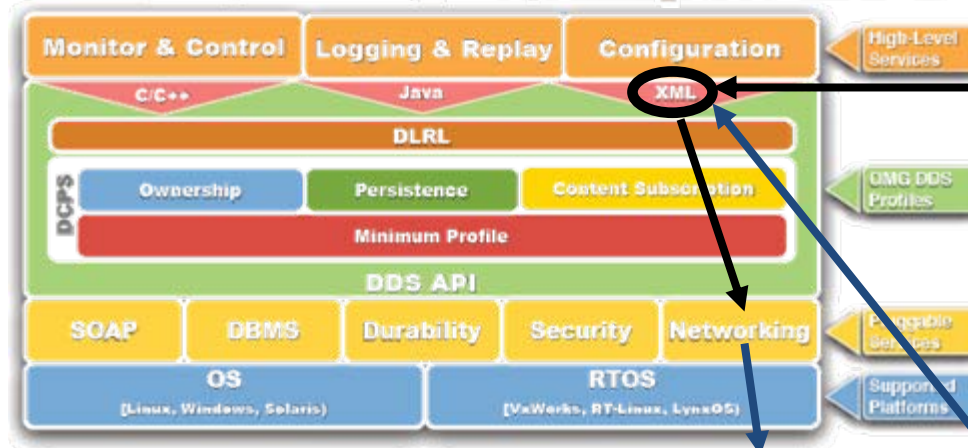


Developed ANT pluggable transport

OpenSplice Pluggable Transport Framework

PrismTech's OpenSplice DDS network plug-in provides:

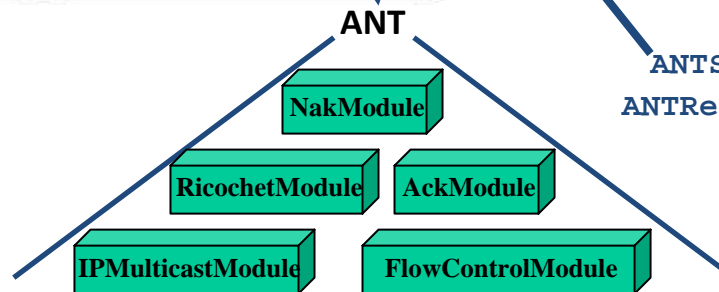
- Efficient C API
- XML configuration file to specify transport functions used



```
SendChannelMessageStart()
ReceiveChannelMessageStart()
.
.
.
```

```
ANTSendChannelMessageStart()
ANTReceiveChannelMessageStart()
.
.
.
```

Developing ANT plug-in



FLEXMAT Test Environment

Using ISISlab/Emulab Environment

- 850 MHz - 3 GHz processors
- Fedora Core 6 w/ RT 11 patches
- Traffic shaping (i.e., network & endhosts)



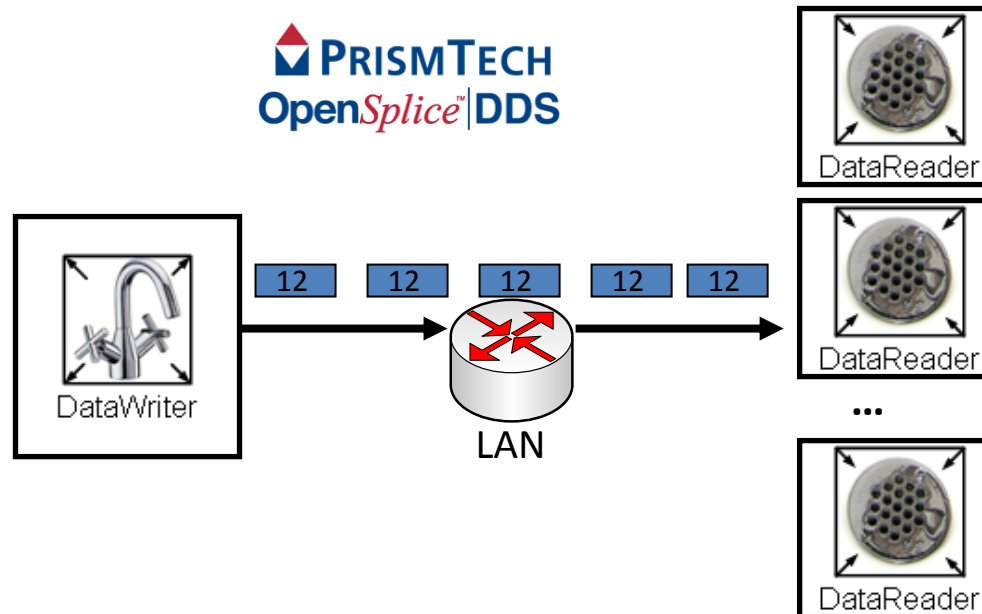
Developed benchmarking infrastructure

- Measures packet latencies, # packets received
- Configurable for # of messages, protocol used, sending rate, # of receivers



Testing scenario

- Using latest OpenDDS (i.e., 1.2.1), OpenSplice (open source)
- 1 data writer, 1 machine, 1 topic
- Variable # of data readers, 1 per machine
- 12 byte data packets, 20K packets sent



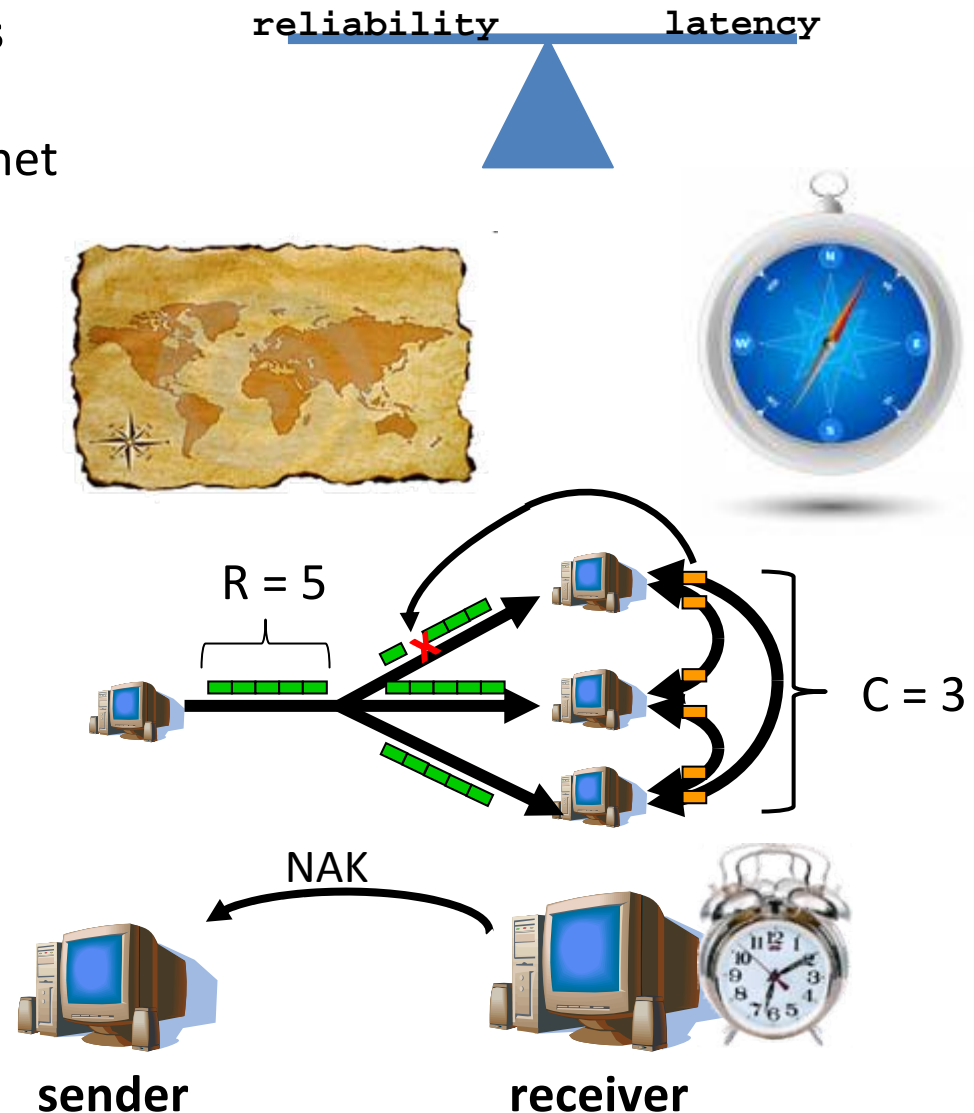
Current Experiments (1/2)

Using standard and custom protocols

- TCP, UDP, IP Mcast, RMcast, ANT Baseline, ANT NAKcast, ANT Ricochet
- Reliability vs. latency

Exploring configuration space

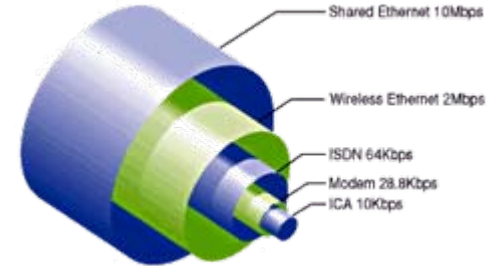
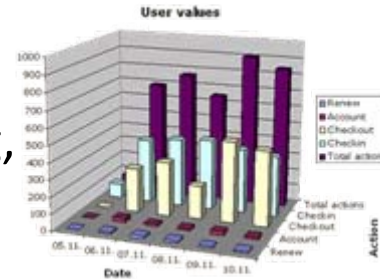
- # of receivers (e.g., 3 to 25)
- % loss in endhosts (e.g., 0 to 10%)
- Sending rate (i.e., 10Hz, 25Hz, 50Hz, 100Hz)
- Ricochet parameter values (e.g., $R=4,8$; $C=3,6$)
- NAKcast timeout values (e.g., 0.05, 0.025 s)



Current Experiments (2/2)

Collecting metrics

- Statistics for latency (e.g., avg, std dev)
- Network bandwidth (e.g., max, min, avg, total bytes)
- Total # of received messages
- ReLate2 metric
 - Includes reliability and latency
 - Assume 10% loss unacceptable (i.e., increases by order of magnitude)

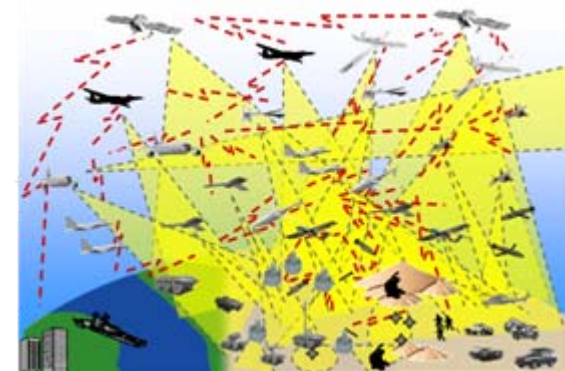


$$ReLate2_p = \frac{\sum_{i=1}^r l_i}{r} \times \left(\frac{t-r}{t} \times 100 + 1 \right)$$

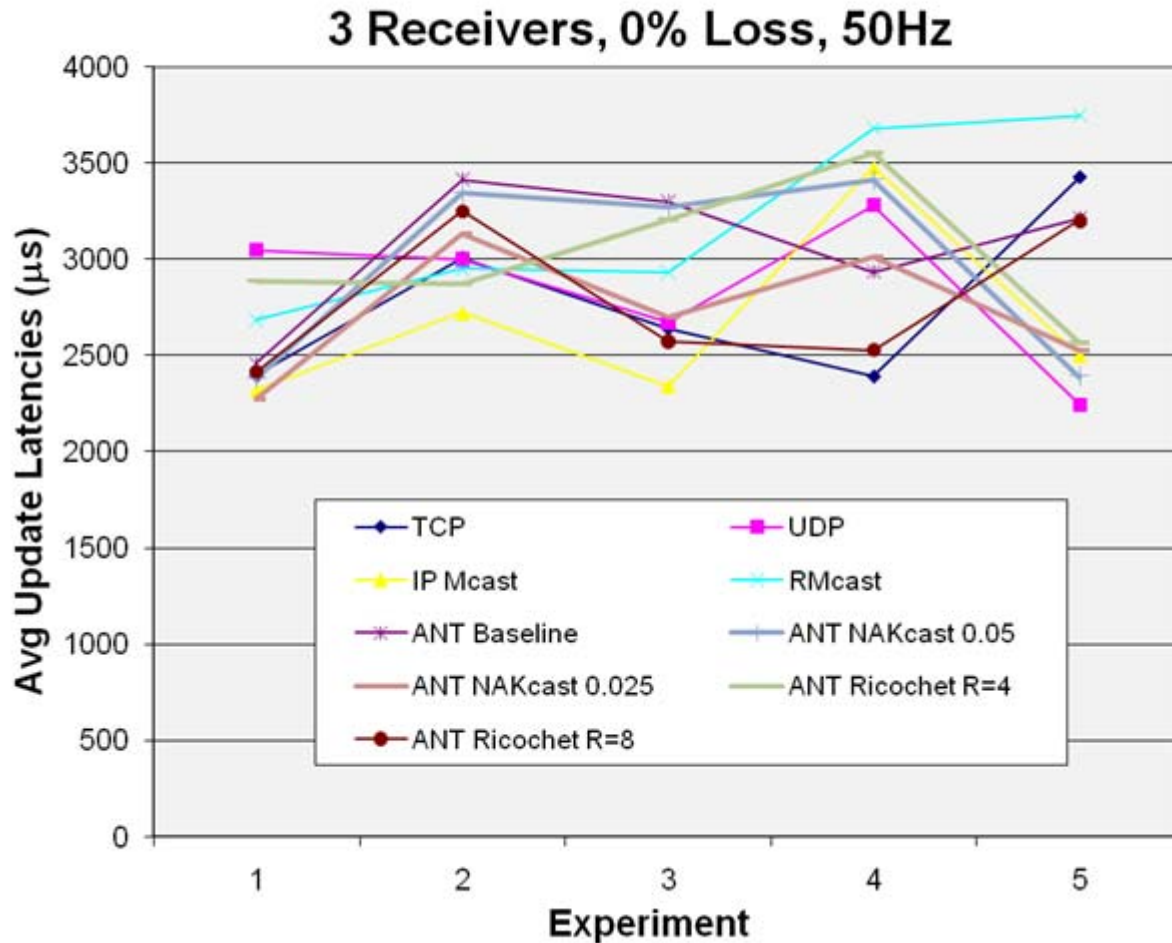
Avg latency **% loss**

Illustrating tradeoffs of protocols

- For dynamic environments
- Leveraging fine-grained QoS and pub-sub abstraction via DDS

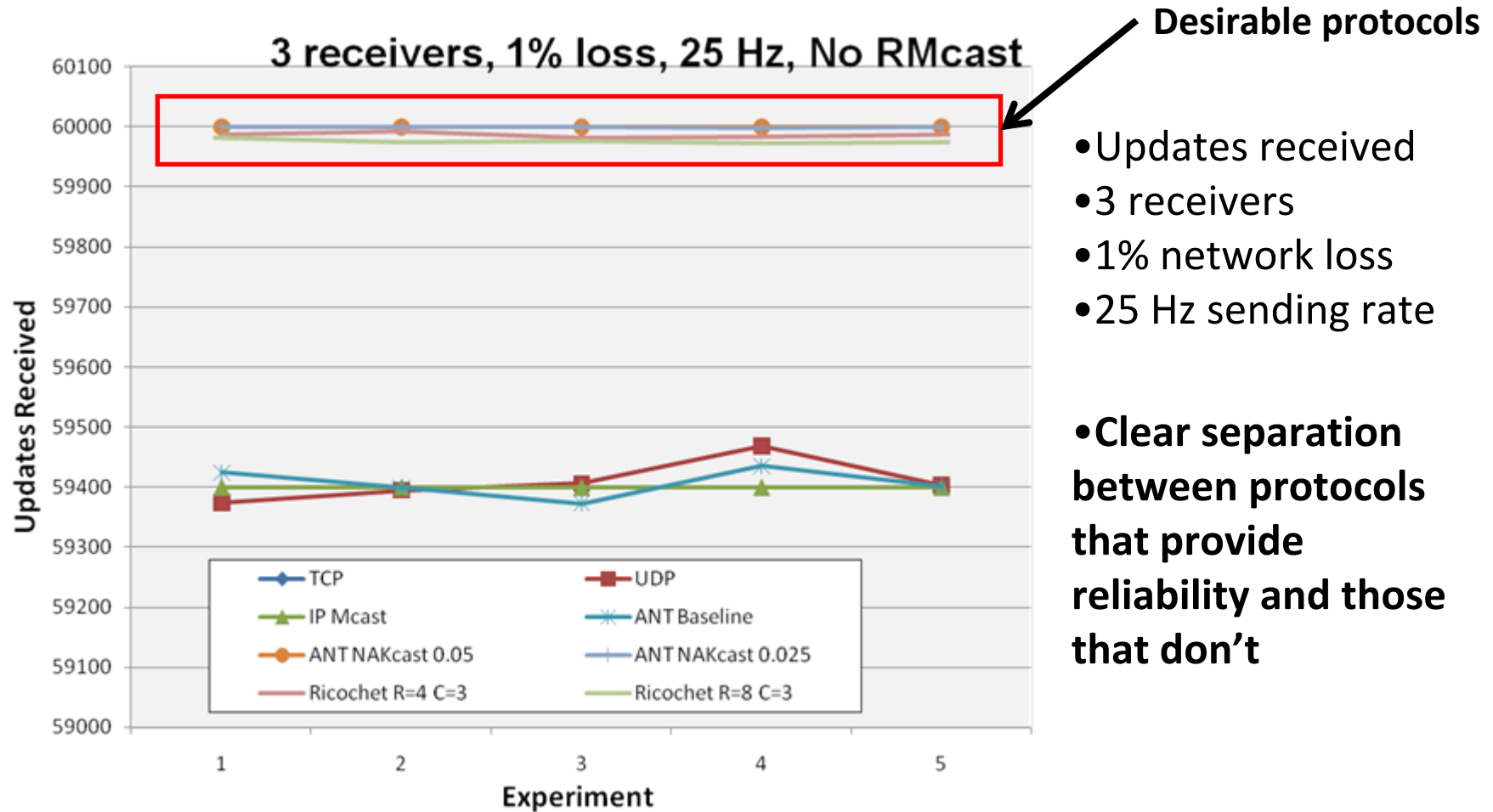


Experiment Results



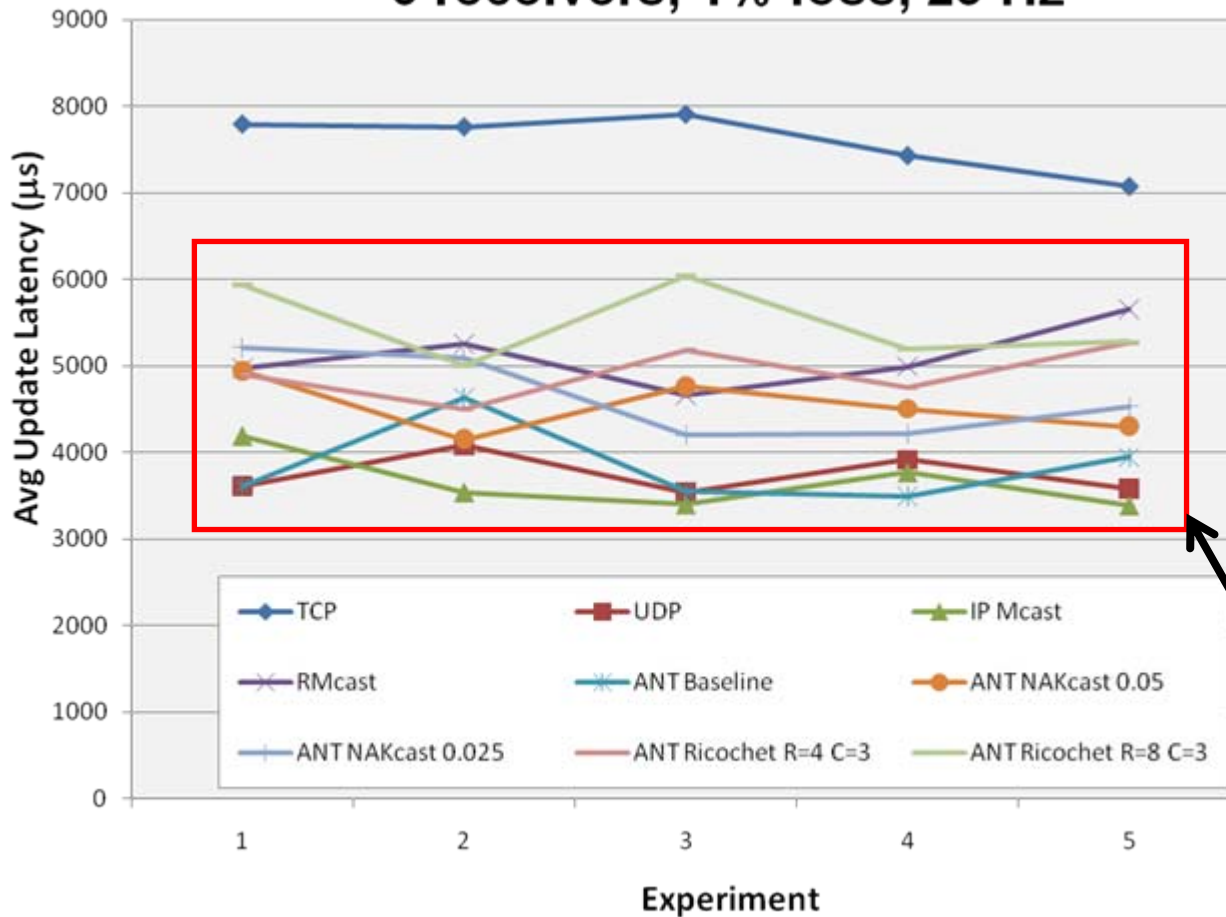
- Avg latency
- 3 receivers
- 0% network loss
- 50 Hz sending rate (same behavior for 25 Hz)
- Lowest latencies from various protocols
- All updates received by all protocols (ReLate2 values equal avg latency)

Experiment Results



Experiment Results

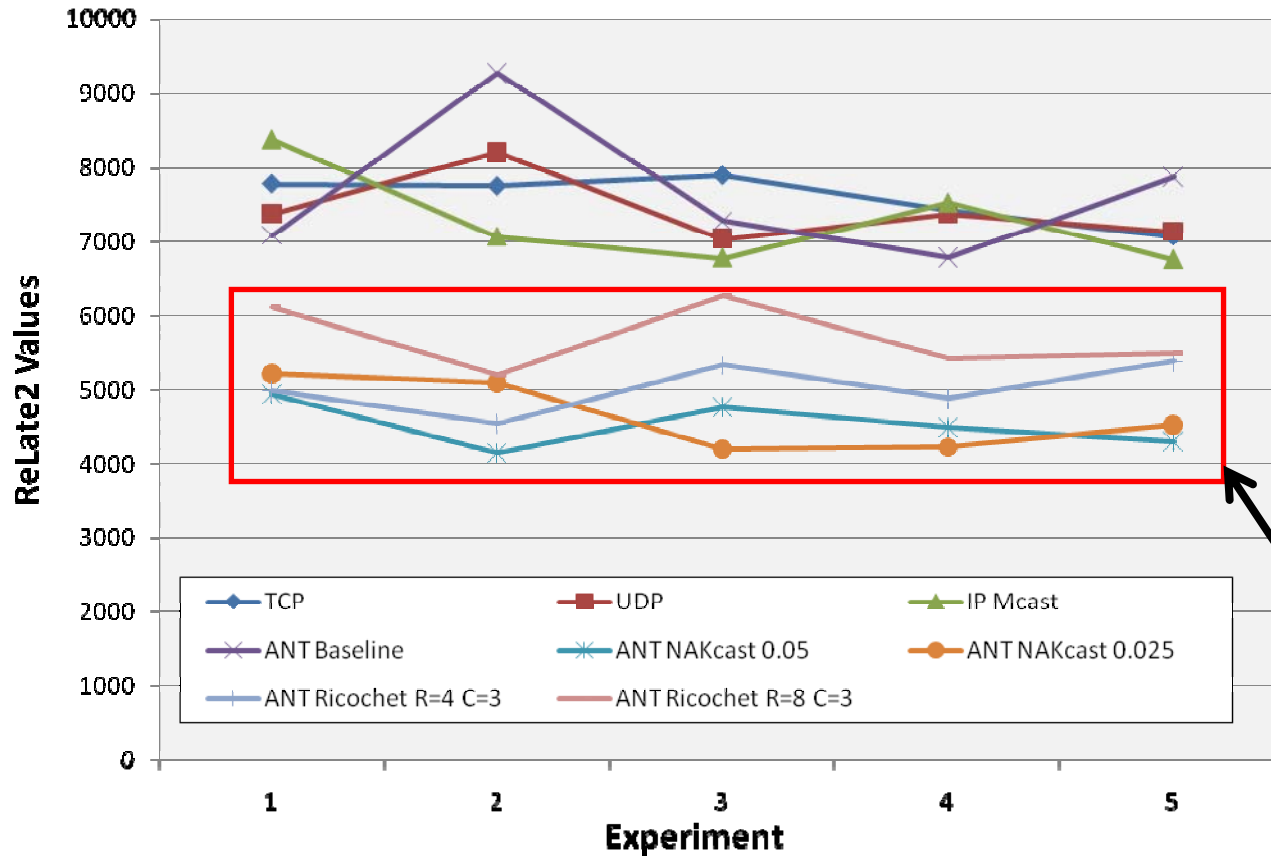
3 receivers, 1% loss, 25 Hz



- Avg latency
 - 3 receivers
 - 1% network loss
 - 25 Hz sending rate
 - **Lowest latencies from protocols w/ no reliability**
 - **Highest latencies from TCP – clear separation**
- Desirable protocols

Experiment Results

3 receivers, 1% loss, 25 Hz



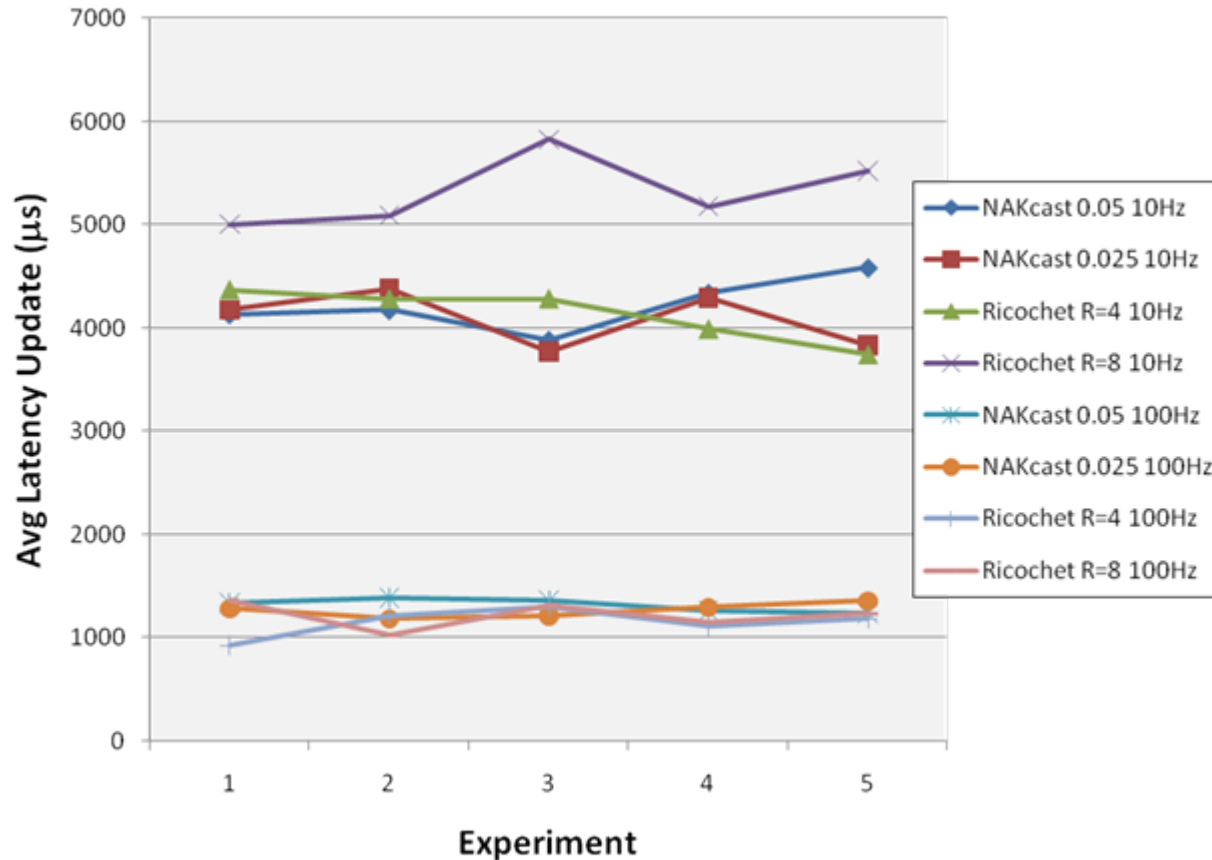
- ReLate2 values
- 3 receivers
- 1% network loss
- 25 Hz sending rate

• Separation between protocols that provide reliability and low latency from those that don't

Desirable protocols

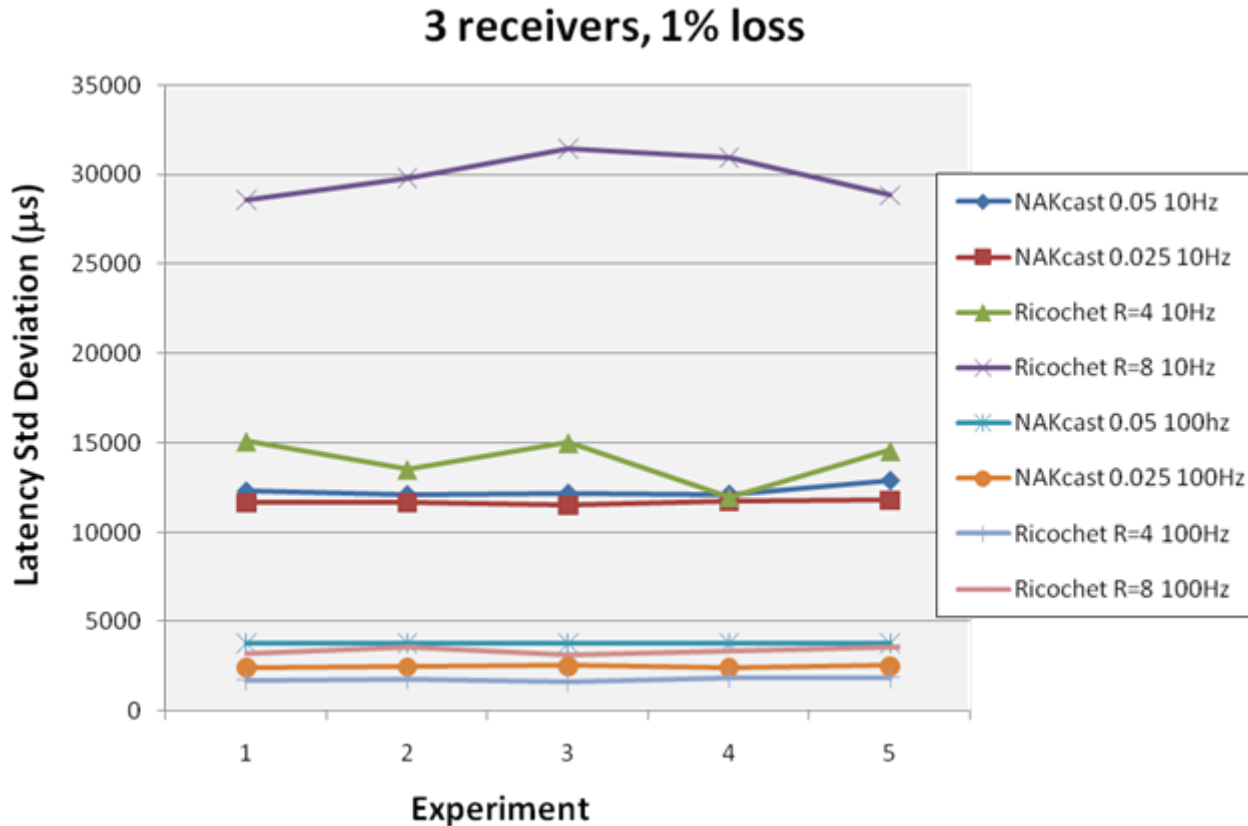
Experiment Results (ANT Protocols)

3 receivers, 1% loss



- **Avg latency**
- 3 receivers
- 1% network loss
- 10, 100 Hz
- **At low rates, NAKcast has most lowest latencies; at high rates, Ricochet R=4**

Experiment Results (ANT Protocols)

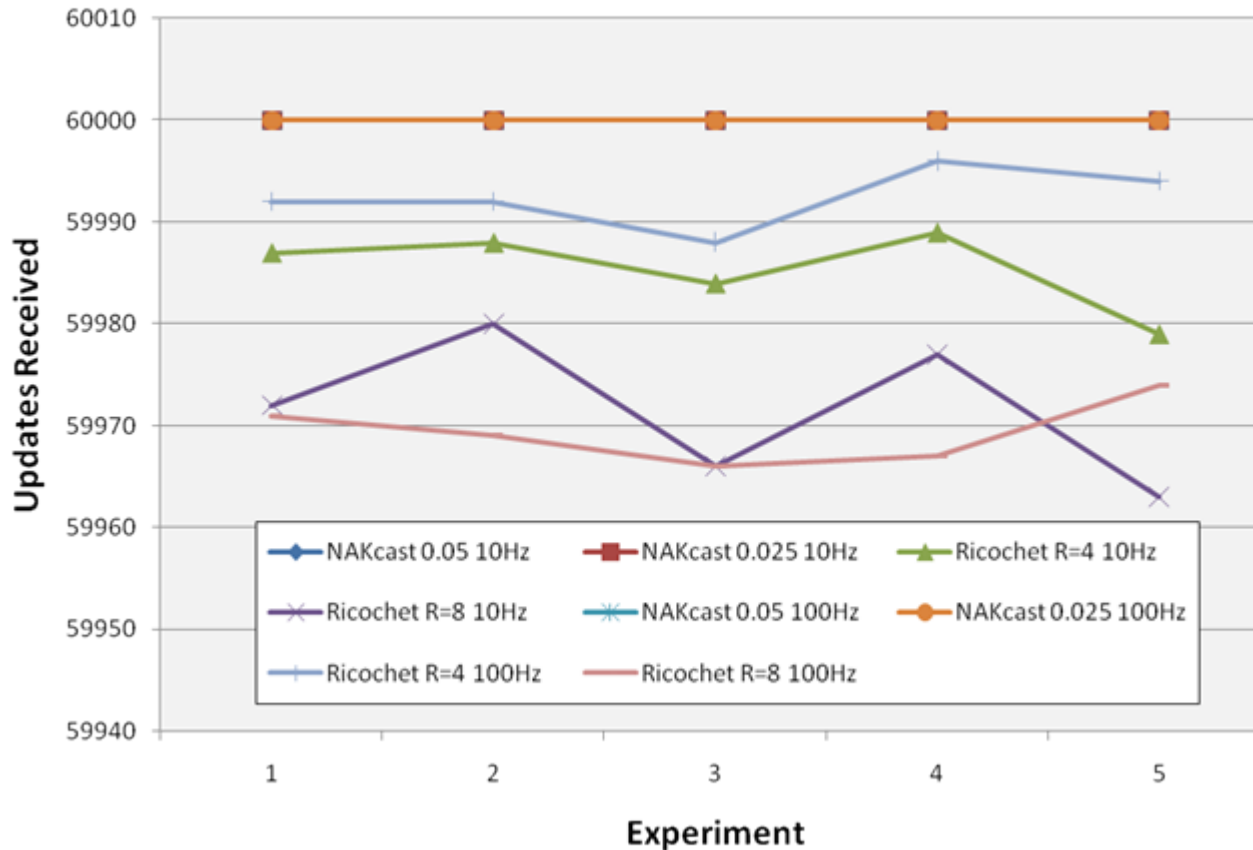


- Std deviation
- 3 receivers
- 1% network loss
- 10, 100 Hz

- **At low rates, NAKcast 0.025 consistently has lowest std deviation; at high rates, Ricochet R=4 is best & Ricochet R=8 beats NAKcast 0.05**

Experiment Results (ANT Protocols)

3 receivers, 1% loss

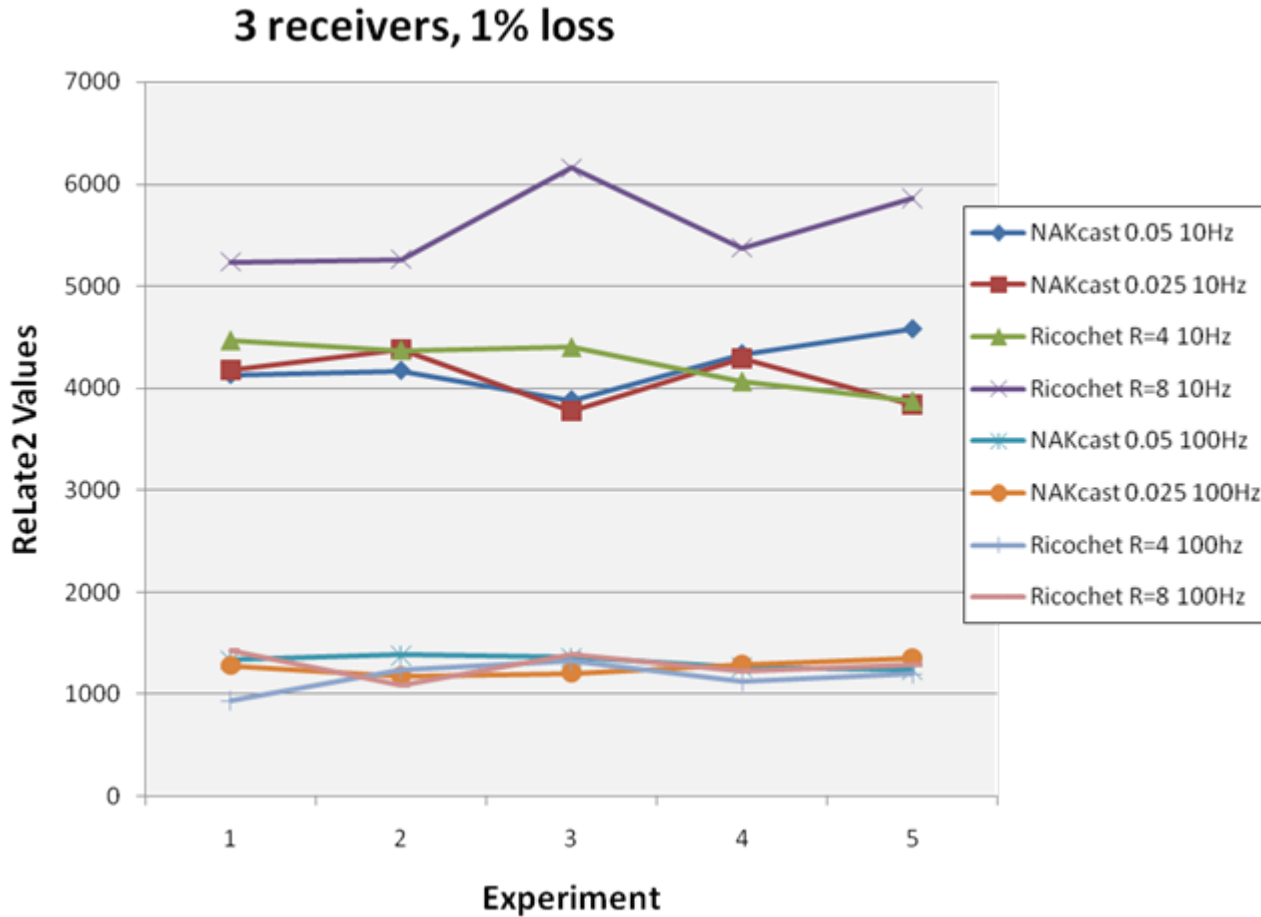


- Updates received

- 3 receivers
- 1% network loss
- 10, 100 Hz

- NAKcast (0.05 and 0.025) consistently get all updates; Ricochet largely unaffected by rate
- Higher Ricochet R value yields higher reliability; more frequent correction data = less chance of loss

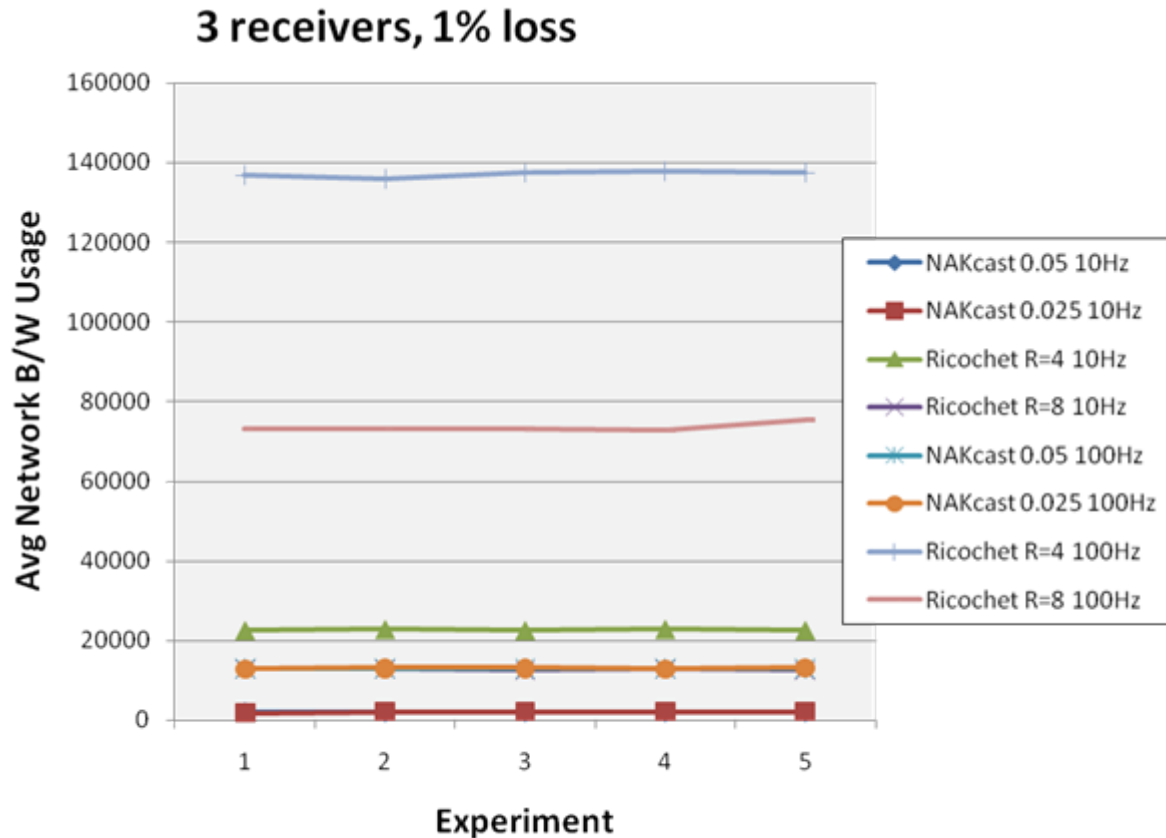
Experiment Results (ANT Protocols)



- **ReLate2 values**
- 3 receivers
- 1% network loss
- 10, 100 Hz

- **At low rates, NAKcast mostly better; at higher rates, Ricochet**

Experiment Results (ANT Protocols)



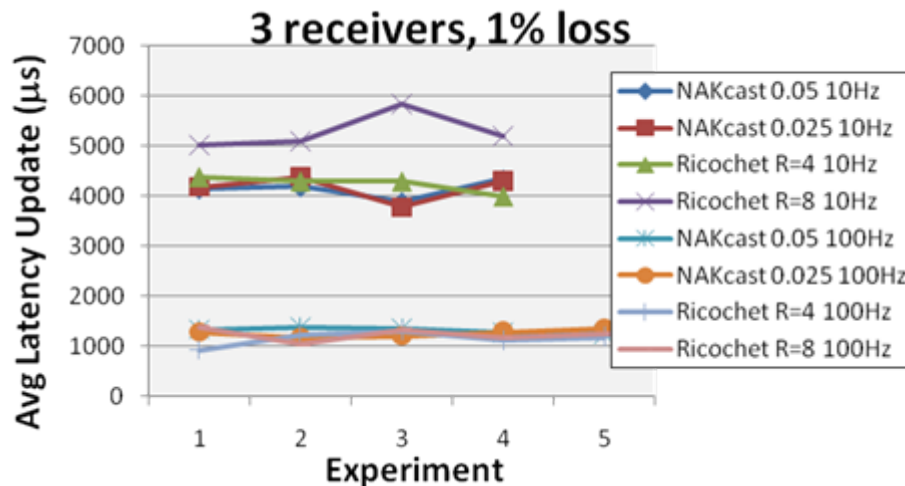
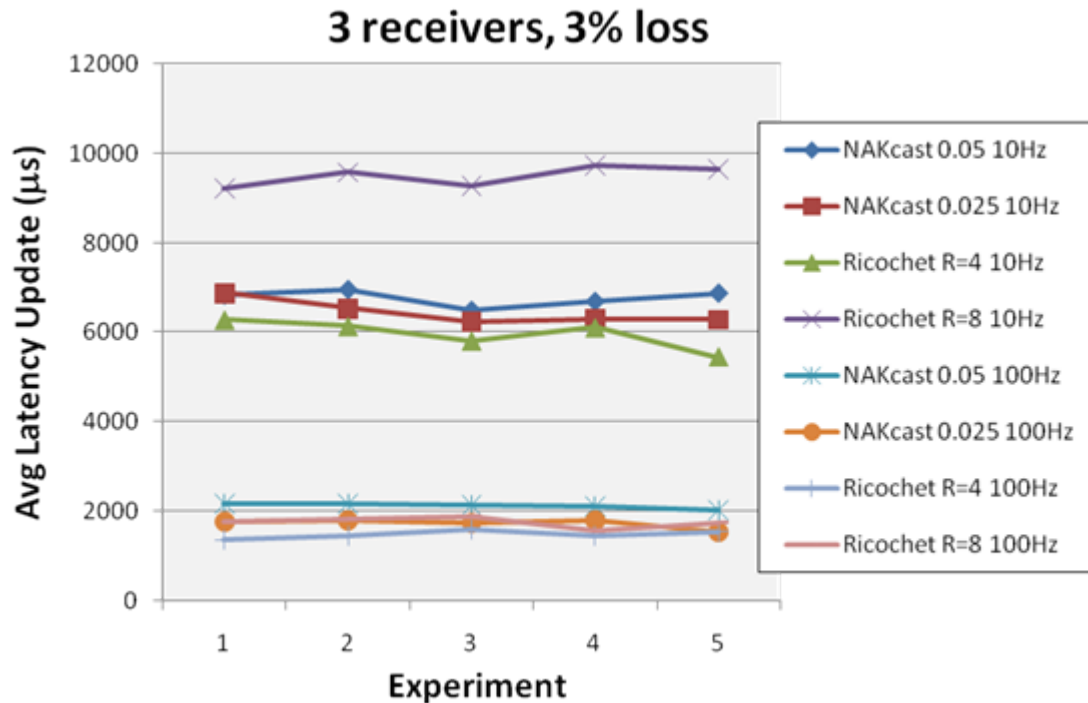
- **Network b/w usage**

- 3 receivers
- 1% network loss
- 10, 100 Hz

- Ricochet uses significantly more b/w, particularly when R=4

- Ricochet R=8 10Hz comparable to NAKcast at 100 Hz

Experiment Results (ANT Protocols)

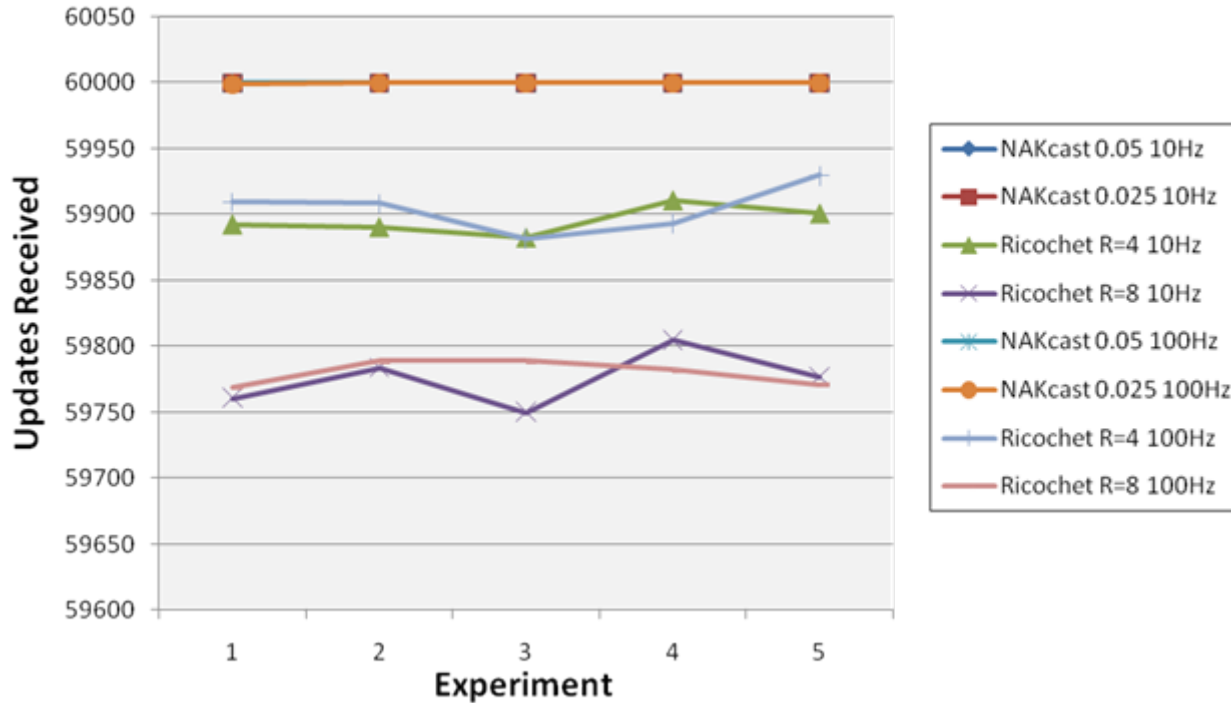


- Avg latency
- 3 receivers
- 3% network loss
- 10, 100 Hz

- At higher % loss Ricochet R=4 has consistently better latency times (cf. 1% loss)

Experiment Results (ANT Protocols)

3 receivers, 3% loss

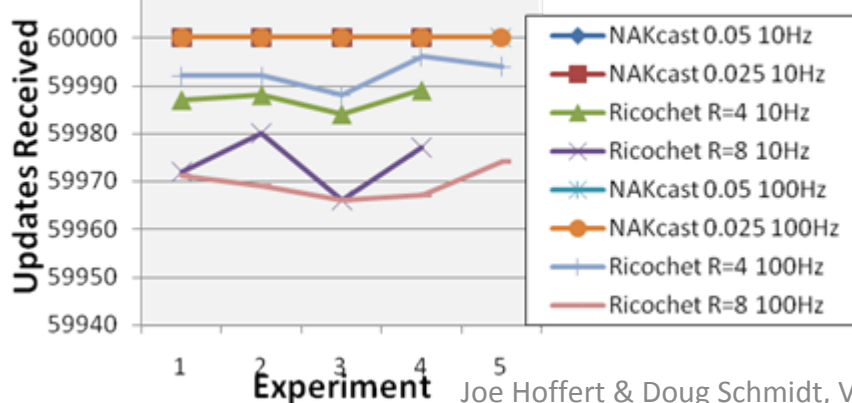


- Updates received

- 3 receivers
- 3% network loss
- 10, 100 Hz

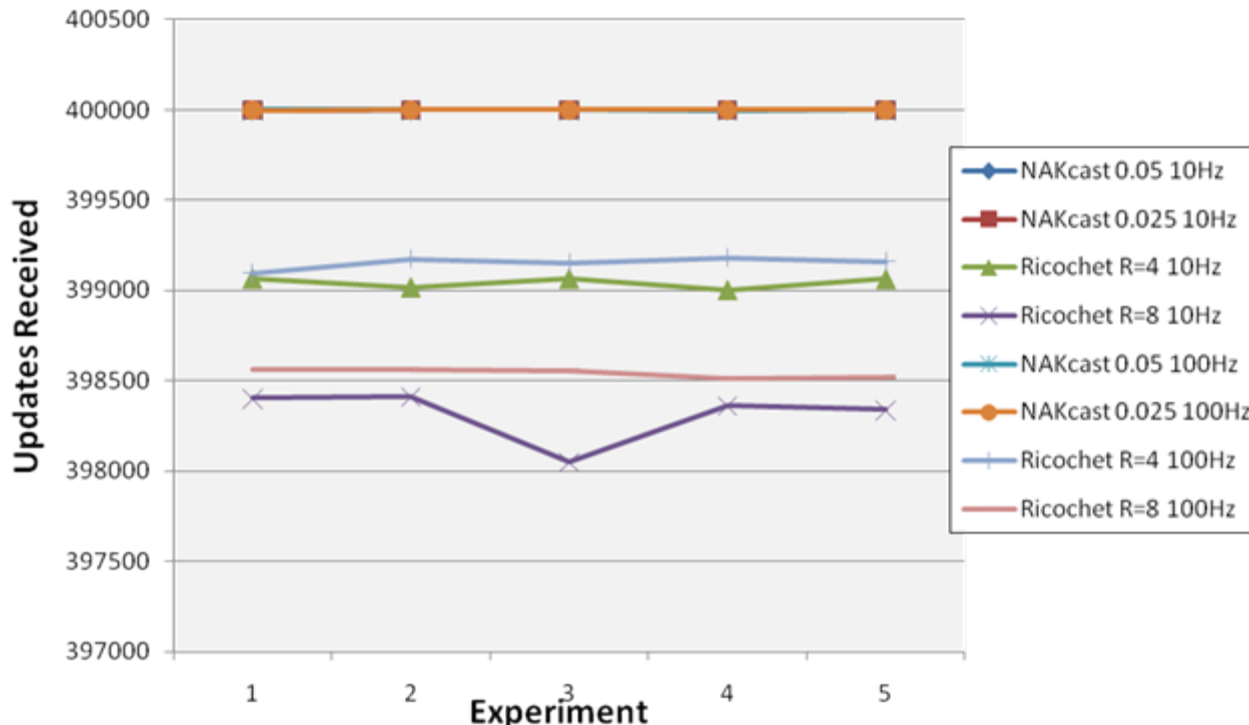
- At higher % loss Ricochet's reliability decreases but still high – 99.57 to 99.88% (cf. 1% loss)

3 receivers, 1% loss

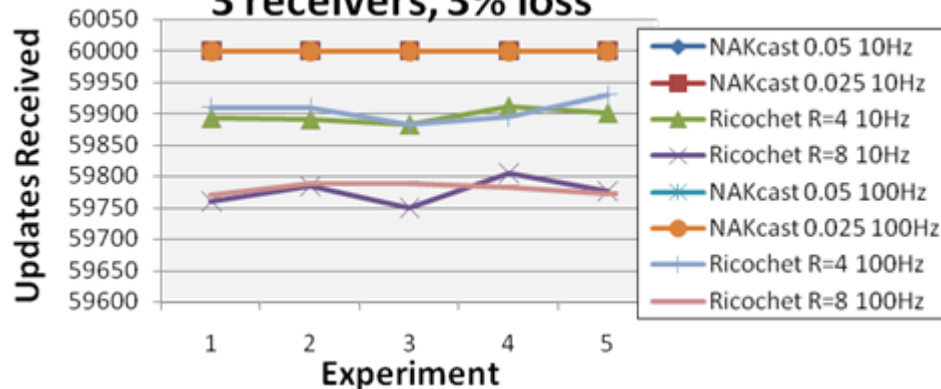


Experiment Results (ANT Protocols)

20 receivers, 3% loss



3 receivers, 3% loss

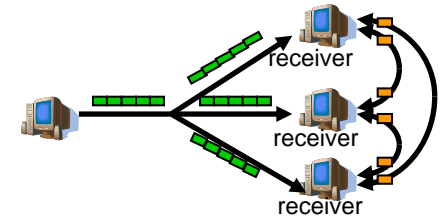
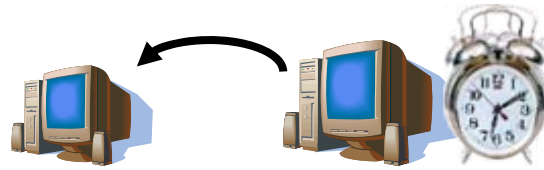


- Updates received
- 20 receivers
- 3% network loss
- 10, 100 Hz
- Ricochet's % loss almost identical to 3 receivers – 99.51 to 99.80% (cf. 99.57 to 99.88%)
- Other data comparable to results from 3 receivers

Concluding Remarks

Evaluation of protocols

- Reliable NAK-based and LEC protocols balance reliability and low latency
- NAK-based -> reliability
- LEC -> latency
- Ricochet consistency:
 - lower latency, std dev at higher rates
 - reliability
 - bandwidth usage (high)



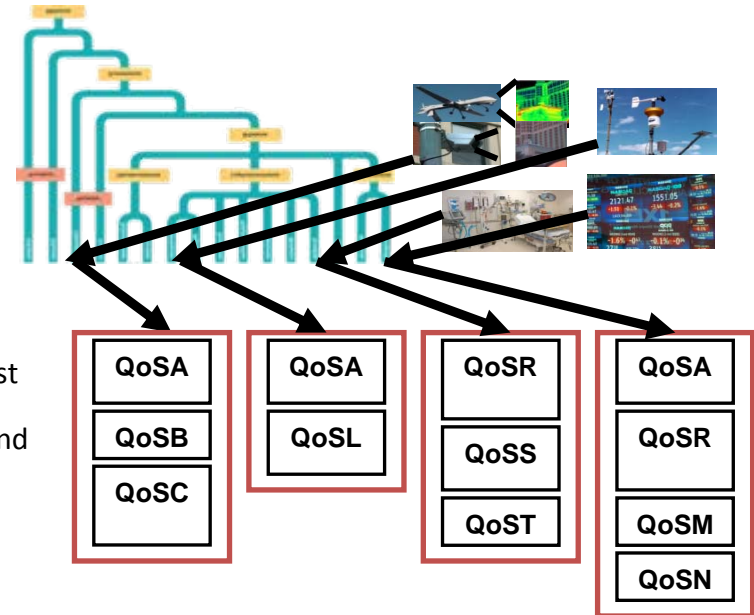
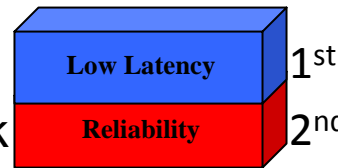
Ricochet



NAKcast

Further research

- Add'l exploration of variability (e.g., NAKcast timeout; Ricochet R, C; latency; jitter; reliability; network usage)
- Management of contentious QoS
 - Hierarchy/prioritization
 - New QoS policies (e.g., network resource limits)
 - Patterns/profiles for application types



<http://www.dre.vanderbilt.edu/~jhoffert/ADAMANT>