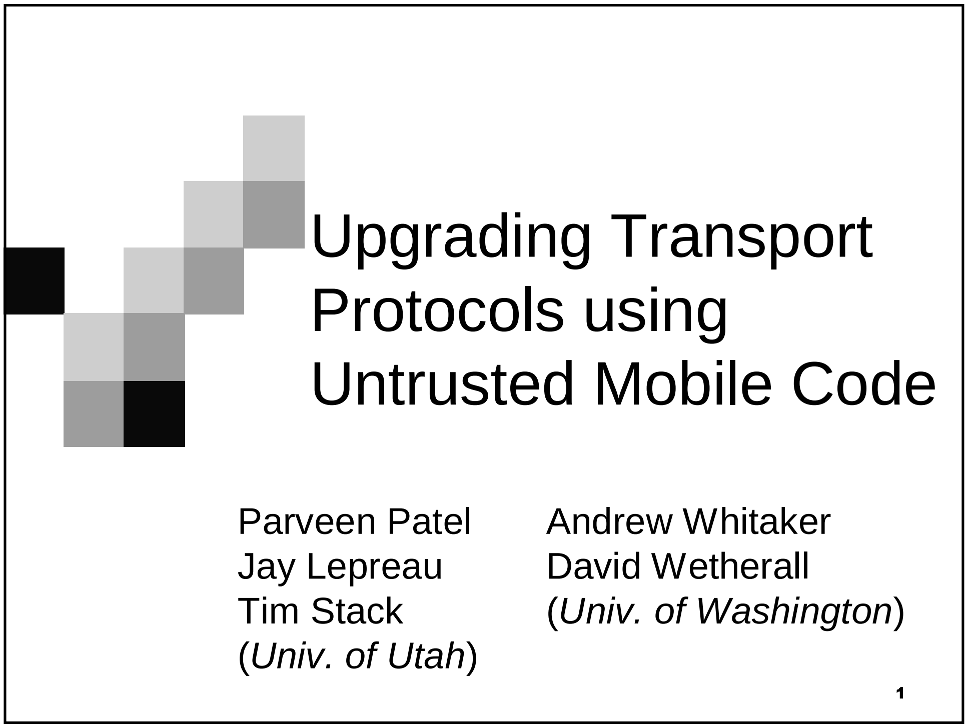
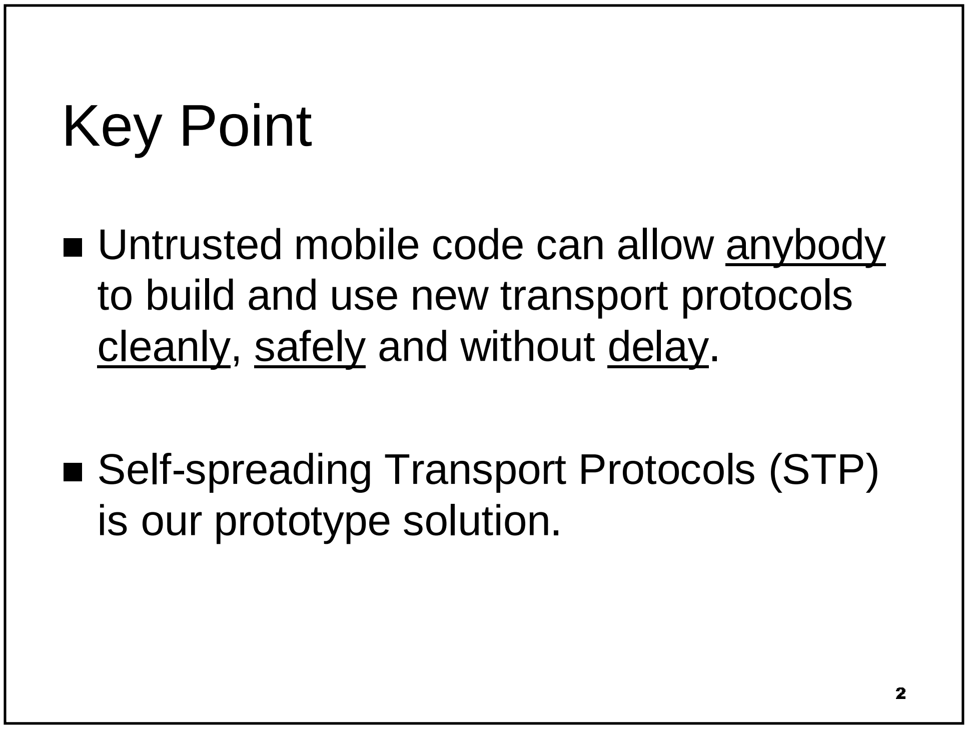


Upgrading Transport Protocols using Untrusted Mobile Code

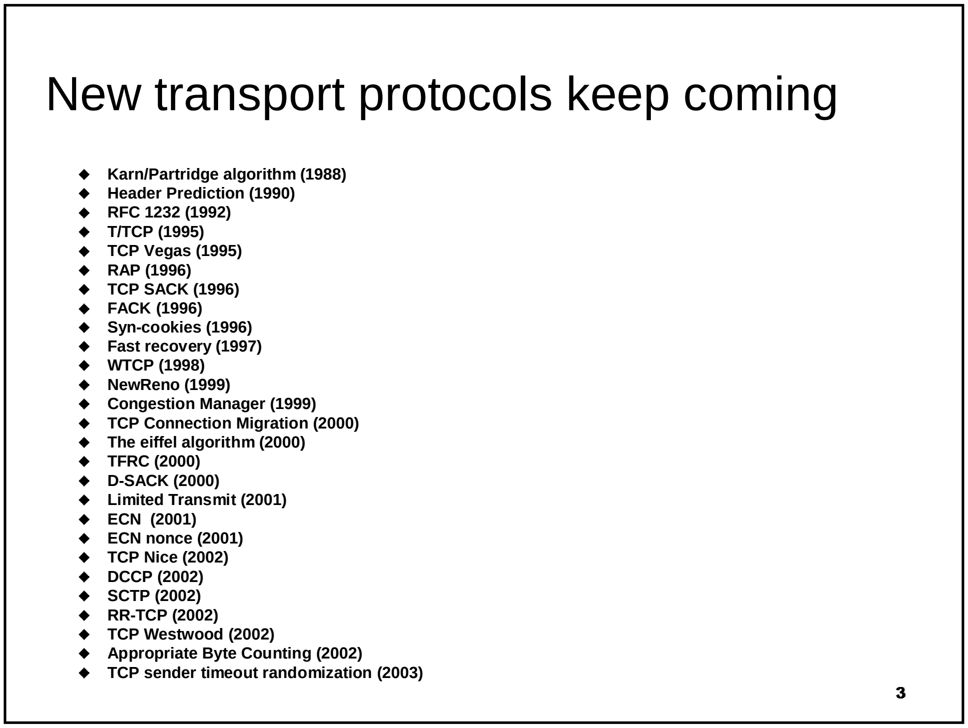
Parveen Patel
Jay Lepreau
Tim Stack
(*Univ. of Utah*)

Andrew Whitaker
David Wetherall
(*Univ. of Washington*)



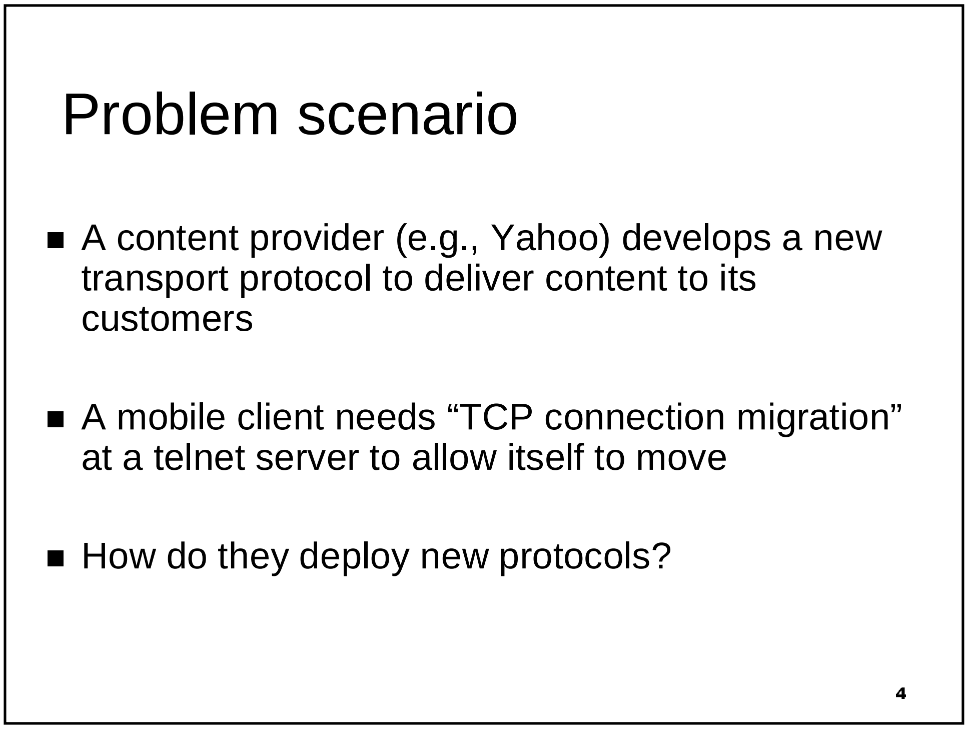
Key Point

- Untrusted mobile code can allow anybody to build and use new transport protocols cleanly, safely and without delay.
- Self-spreading Transport Protocols (STP) is our prototype solution.



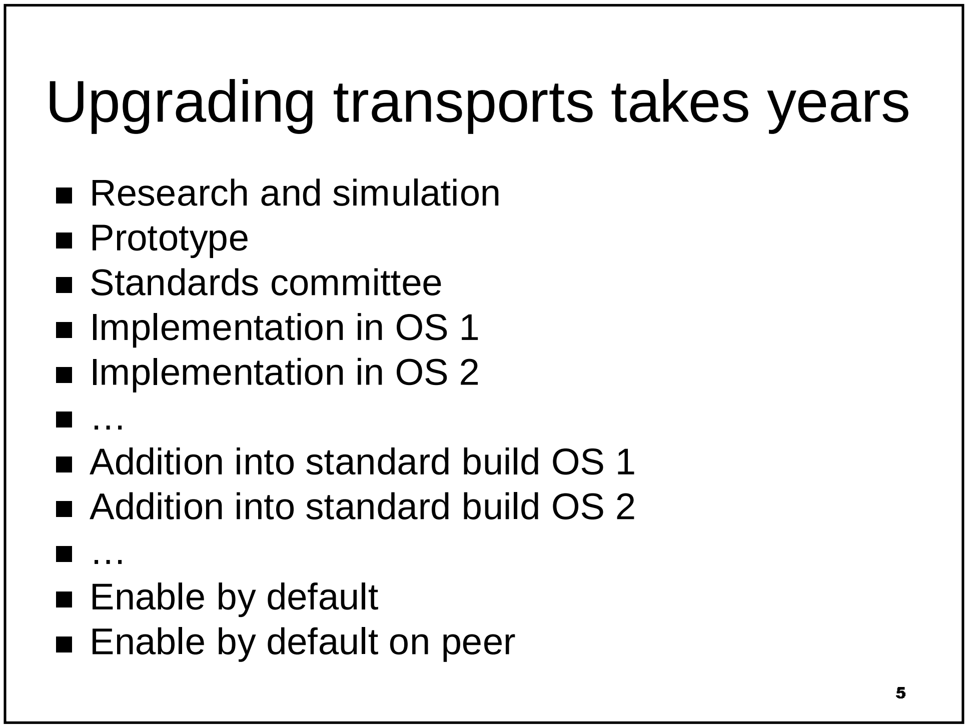
New transport protocols keep coming

- Karn/Partridge algorithm (1988)
- Header Prediction (1990)
- RFC 1232 (1992)
- T/TCP (1995)
- TCP Vegas (1995)
- RAP (1996)
- TCP SACK (1996)
- FACK (1996)
- Syn-cookies (1996)
- Fast recovery (1997)
- WTCP (1998)
- NewReno (1999)
- Congestion Manager (1999)
- TCP Connection Migration (2000)
- The eiffel algorithm (2000)
- TFRC (2000)
- D-SACK (2000)
- Limited Transmit (2001)
- ECN (2001)
- ECN nonce (2001)
- TCP Nice (2002)
- DCCP (2002)
- SCTP (2002)
- RR-TCP (2002)
- TCP Westwood (2002)
- Appropriate Byte Counting (2002)
- TCP sender timeout randomization (2003)



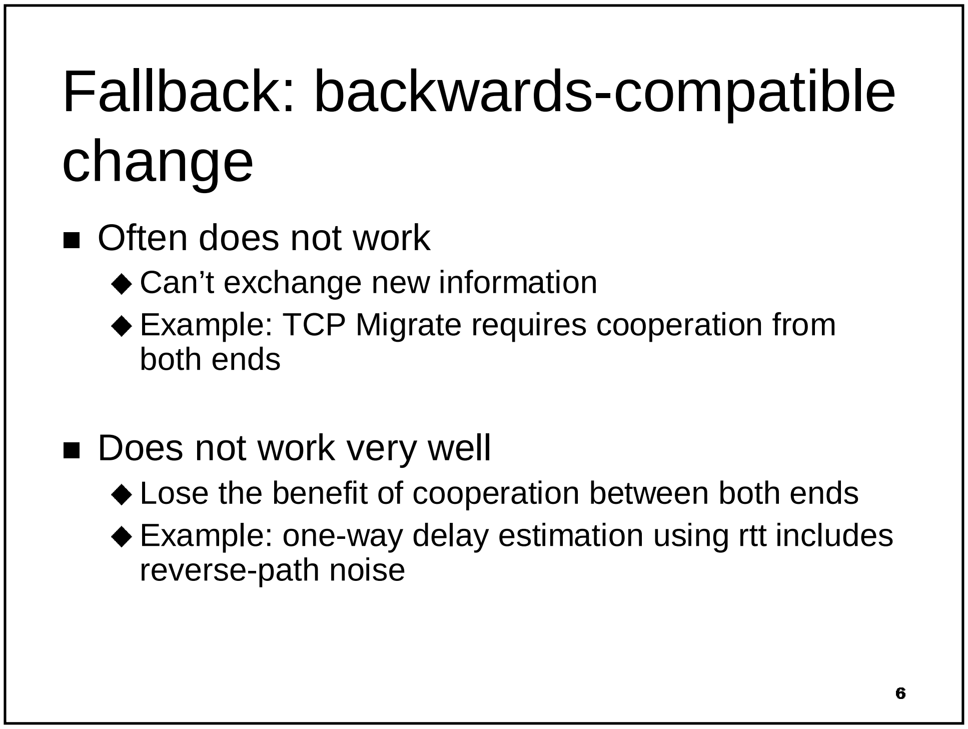
Problem scenario

- A content provider (e.g., Yahoo) develops a new transport protocol to deliver content to its customers
- A mobile client needs “TCP connection migration” at a telnet server to allow itself to move
- How do they deploy new protocols?



Upgrading transports takes years

- Research and simulation
- Prototype
- Standards committee
- Implementation in OS 1
- Implementation in OS 2
- …
- Addition into standard build OS 1
- Addition into standard build OS 2
- …
- Enable by default
- Enable by default on peer

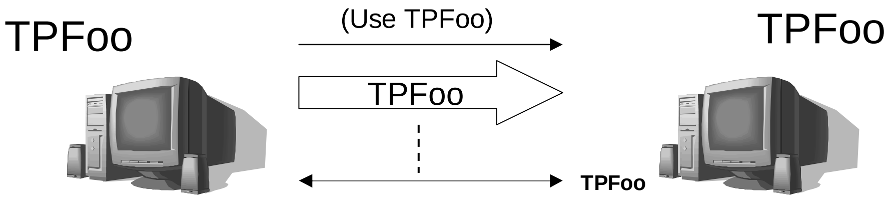


Fallback: backwards-compatible change

- Often does not work
  - Can’t exchange new information
  - Example: TCP Migrate requires cooperation from both ends
- Does not work very well
  - Lose the benefit of cooperation between both ends
  - Example: one-way delay estimation using rtt includes reverse-path noise

Solution: STP

- Host can upgrade its connection peer with new transports by sending untrusted code



Self-spreading Transport Protocols

Upgrading with STP is faster

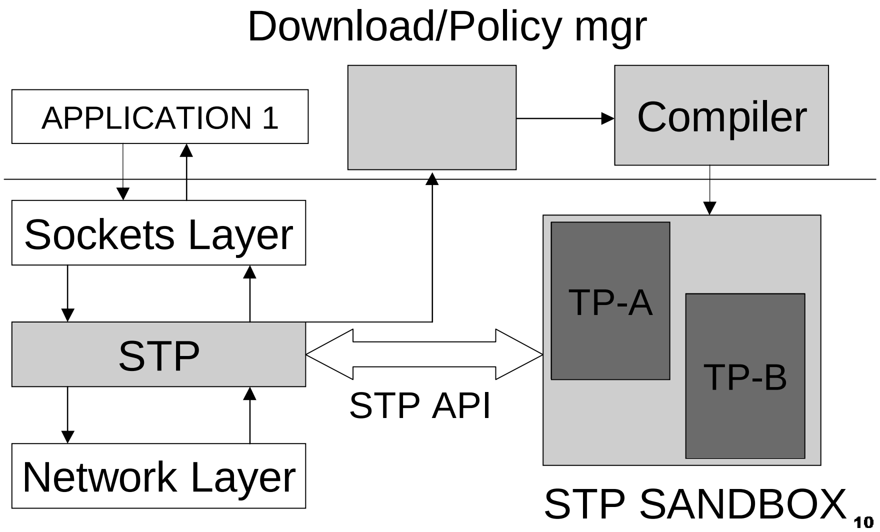
- Research and simulation
- Prototype
- Standards committee
- Implementation to the STP API
- Implementation in OS 1
- Implementation in OS 2
- …
- Addition into standard build OS 1
- Addition into standard build OS 2
- …
- Enable by default
- Enable by default on peer

STP Challenges

1. Network safety – should not hog bandwidth or attack other nodes
2. Host safety – must isolate and limit resource consumption
3. Performance – should not undermine improvement due to extensions

9

STP Design



10

1. Network safety TCP background

- TCP-friendliness is well-defined [SIGCOMM '98]

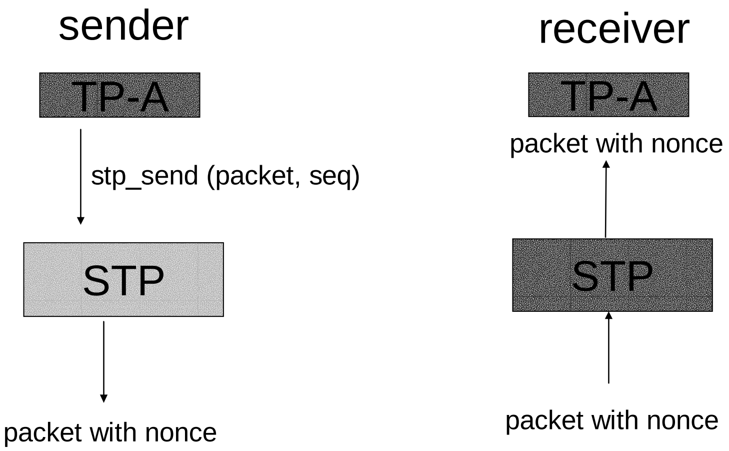
$$\text{Rate} = \frac{1}{R*\surd(2 * L/3) + (t\_RTO*3* \surd(3*L/8)*L*(1+32+L^{2}))}$$

R = Round-trip time, L = Loss-rate

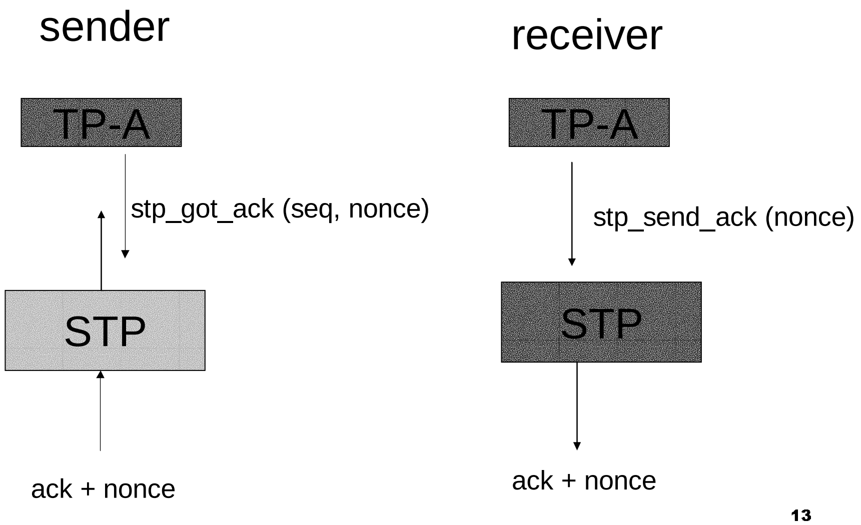
- TCP sending speed governed by inflow of acks from receiver. Prevent a TCP receiver from faking acks (hiding loss) by requiring it to echo a nonce. [ICNP'01]

Loss Detection in STP

Through the design of its API, STP enforces loss detection that is *independent* of transport protocol header formats.



Loss Detection in STP



sender

TP-A

stp_got_ack (seq, nonce)

STP

ack + nonce

receiver

TP-A

stp_send_ack (nonce)

STP

ack + nonce

2. Host safety

- Constrained domain: no shared state between transports
  - Makes resource accounting straightforward
  - Makes termination tractable
- Memory safety: type-safety of Cyclone [PLDI '02]
- CPU timer-based CPU resource protection

3. Performance

- Connections proceed without delays
  - Code is downloaded out of the critical path
  - Benefits later connections
  - Exploits communication pattern of today’s Internet
- Efficient to interface C with Cyclone
  - Share data between the kernel and Cyclone code
  - Not necessary to use garbage collection

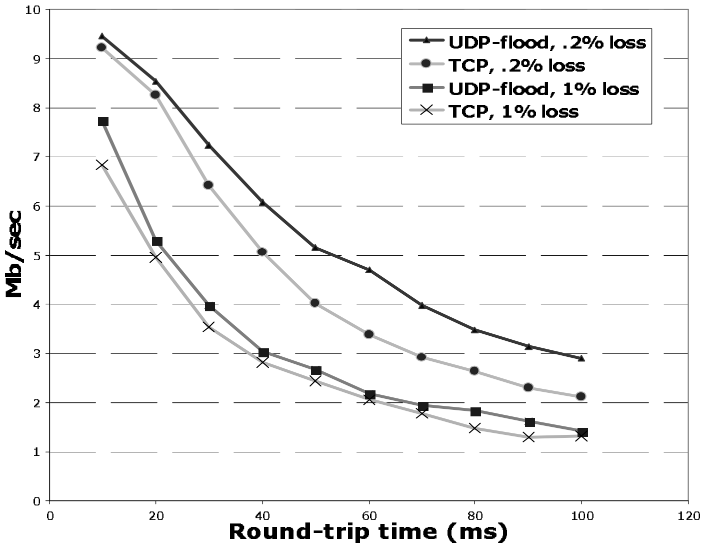
Implementation

- Prototype in FreeBSD 4.7
- Ported UDP-Flood, TCP NewReno and TCP SACK to the STP API

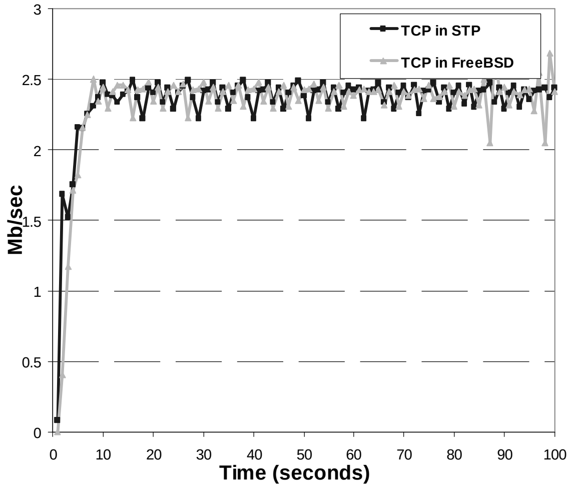
Evaluation

- Network Safety
- Overall Performance
- CPU Overhead
- Transport Experience

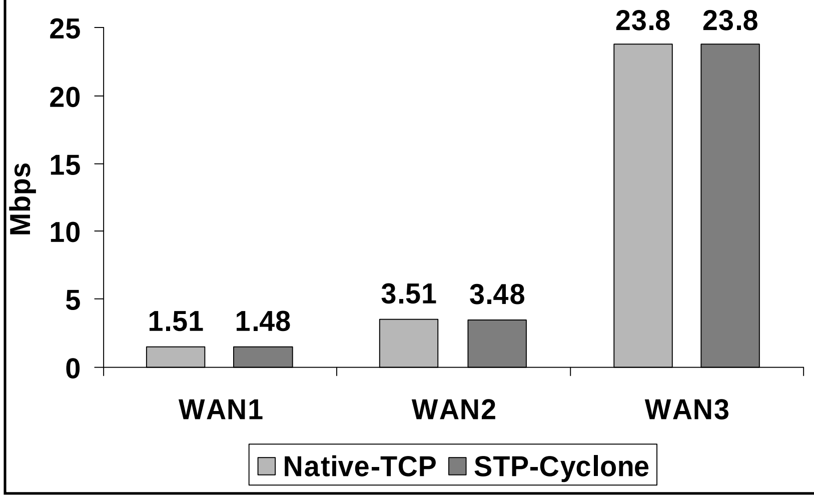
STP enforces TCP-friendliness



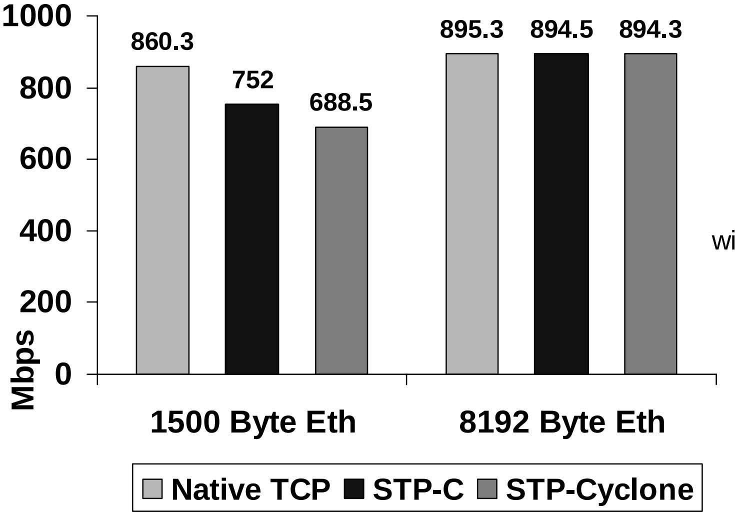
STP does not restrict TCP



STP is as fast as TCP for Internet-like paths



STP transports achieve gigabit speed

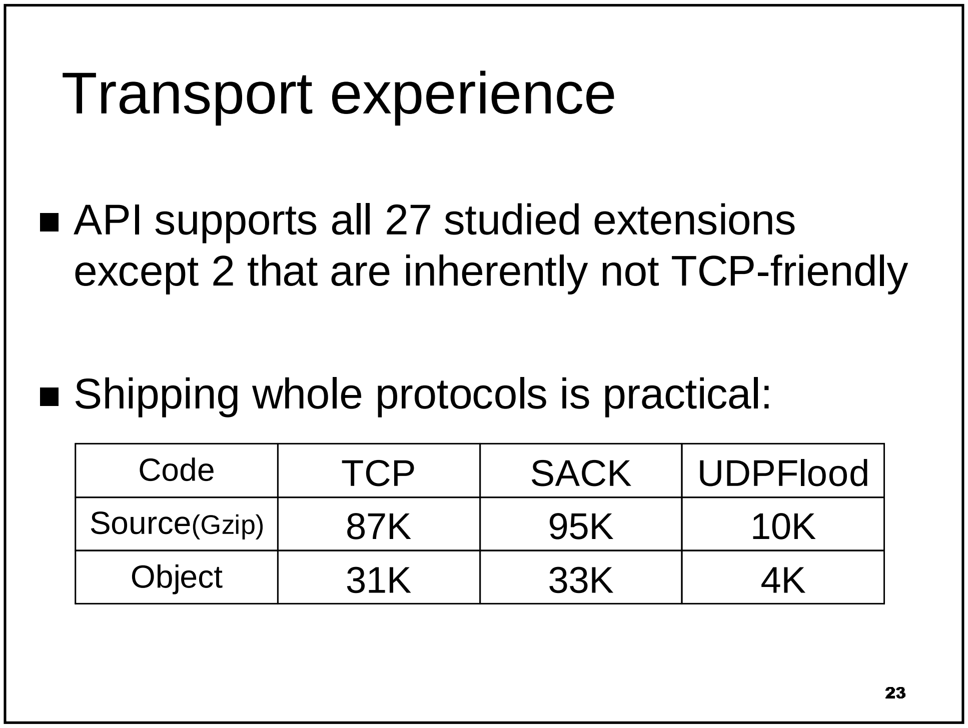


2GHz machine
with fast PCI bus

CPU utilization (gigabit link)

TCP Version	FreeBSD	STP-C (ratio to BSD)	STP-Cyclone (ratio to BSD)
Sender	59%	59% (1.01)	73% (1.24)
Receiver	48%	61% (1.29)	73% (1.54)

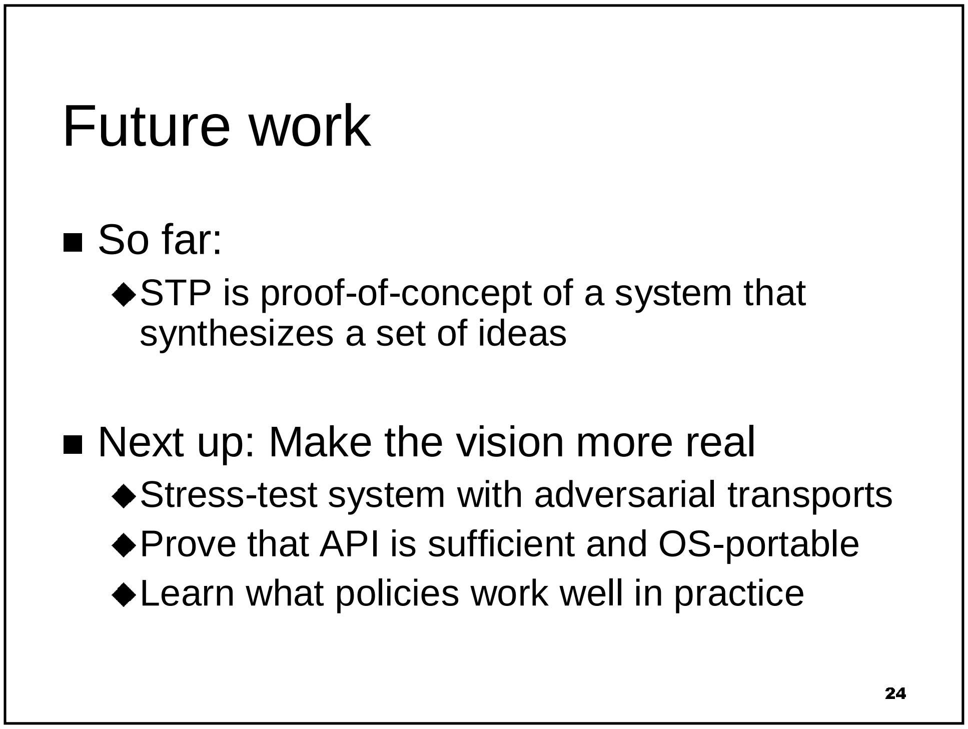
- Overhead inherent in Cyclone’s type-safety (bounds/null checks) is low: 6%
- Suspect most of overhead due to marshaling that will be straightforward to optimize in newer version of compiler.



Transport experience

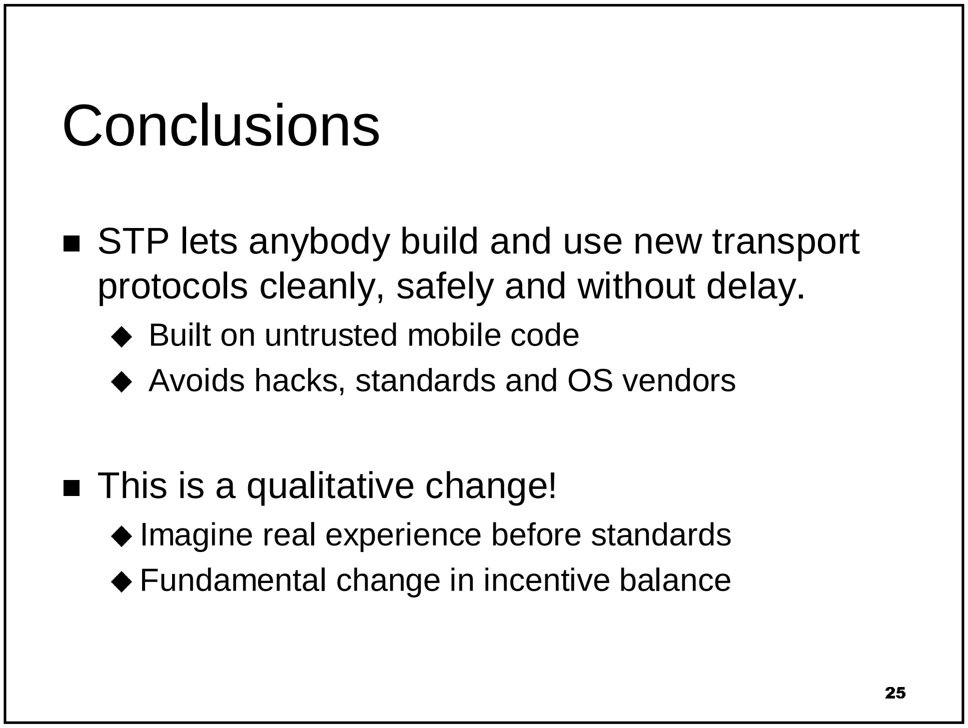
- API supports all 27 studied extensions except 2 that are inherently not TCP-friendly
- Shipping whole protocols is practical:

Code	TCP	SACK	UDPFlood
Source(Gzip)	87K	95K	10K
Object	31K	33K	4K



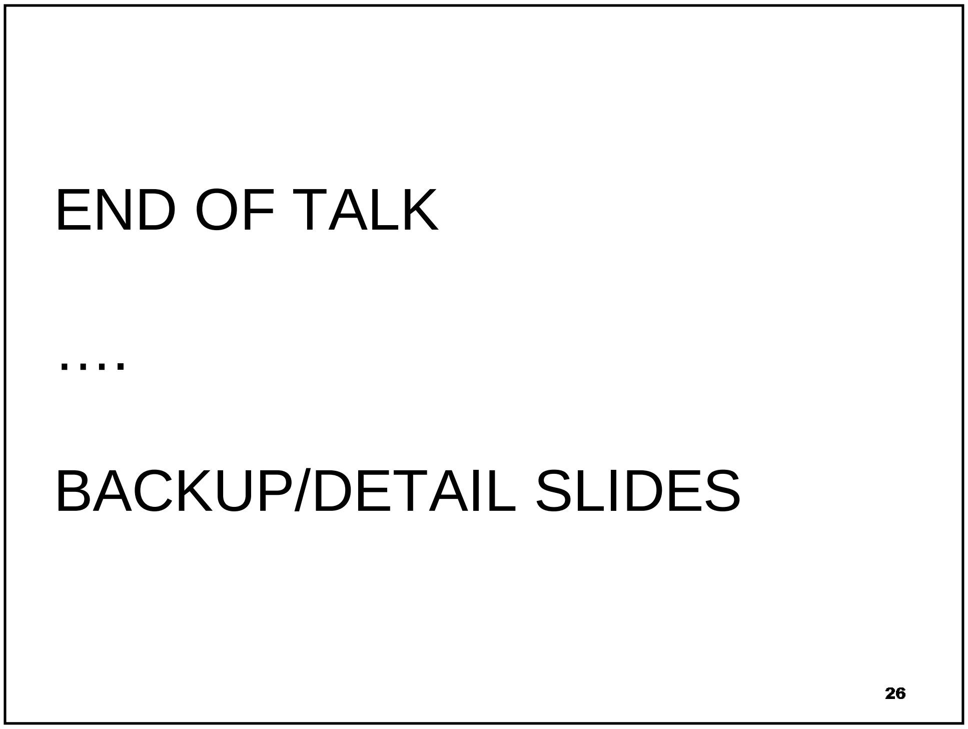
Future work

- So far:
  - ◆STP is proof-of-concept of a system that synthesizes a set of ideas
- Next up: Make the vision more real
  - ◆Stress-test system with adversarial transports
  - ◆Prove that API is sufficient and OS-portable
  - ◆Learn what policies work well in practice



Conclusions

- STP lets anybody build and use new transport protocols cleanly, safely and without delay.
  - Built on untrusted mobile code
  - Avoids hacks, standards and OS vendors
- This is a qualitative change!
  - Imagine real experience before standards
  - Fundamental change in incentive balance



END OF TALK

….

BACKUP/DETAIL SLIDES