"There are three kinds of lies: Lies, Damn Lies, and Benchmarks."

TIPC Analyse und Optimierung

Florian Westphal

13. Juli 2007

1024D/F260502D <fw@strlen.de>
1C81 1AD5 EA8F 3047 7555
E8EE 5E2F DA6C F260 502D

Themenübersicht

- TIPC-Einführung
  - Motivation, Konzepte, Eigenschaften
  - Netzaufbau und Adressierung
- Untersuchung
  - Test-Cases
  - Optimierungen
- Fazit

TIPC: Einführung

Transparent Inter-Process Communication

- Entwickelt bei Ericsson
- GPL/BSD - Lizenz (Dez. 2000)
- Implementation (derzeit) für Linux und VxWorks verfügbar
- Seit 2.6.16 Teil des Linux Kernels
- BSD-Socket-Schicht Integration (`AF_TIPC`)
- Weiterenwicklung durch Multicore Association/TIPC WG
- aktueller Draft: Mai 2006

TIPC: Motivation und Ziele [JM04]

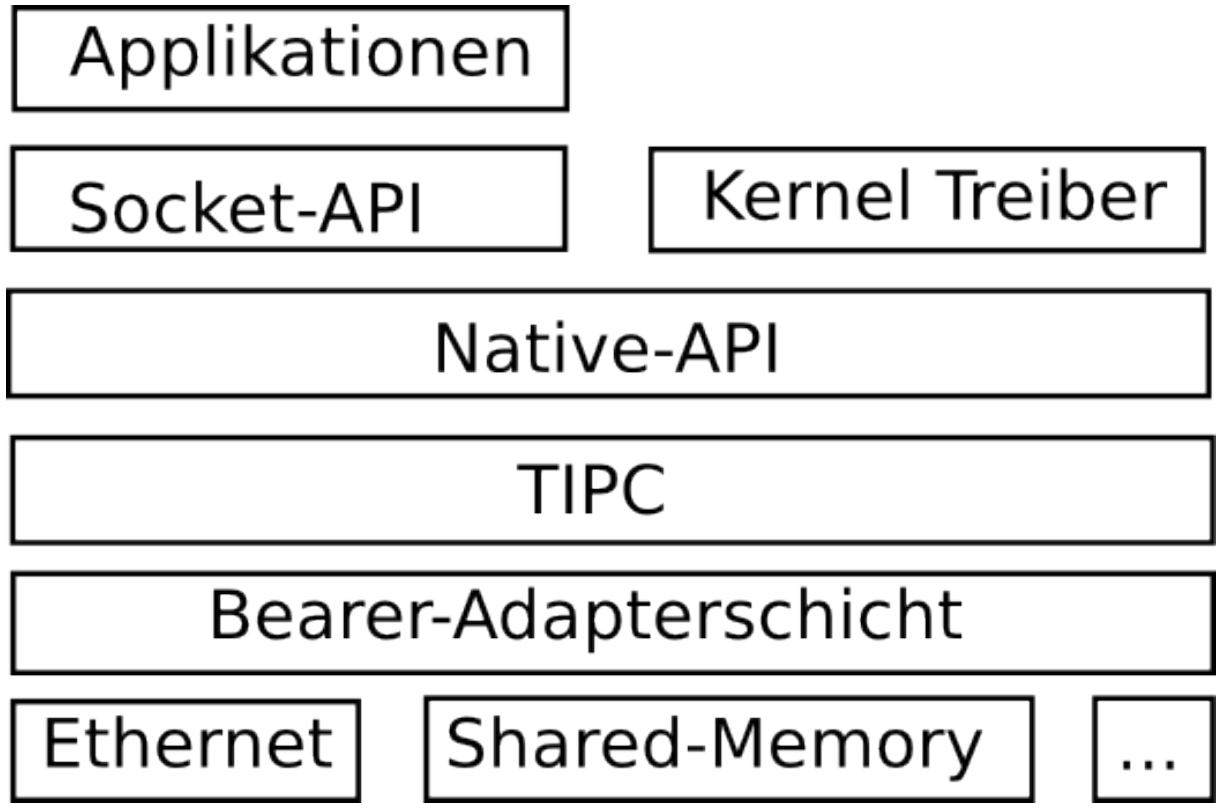
- Effiziente Kommunikation innerhalb von (lose gekoppelten) Clustern
  - einzelne Knoten können Ausfallen, Knoten werden hinzugefügt
  - Dienste werden auf andere Knoten migriert (Load-Balancing)
- $\longrightarrow$ Adresstranzparenz
- $\longrightarrow$ schnelle Fehlererkennung
- TCP, UDP(-lite), SCTP, DCCP, etc. decken nicht alle Anforderungen ab (bzw. haben andere Ziele)

TIPC: Design [JM04]

Annahmen:

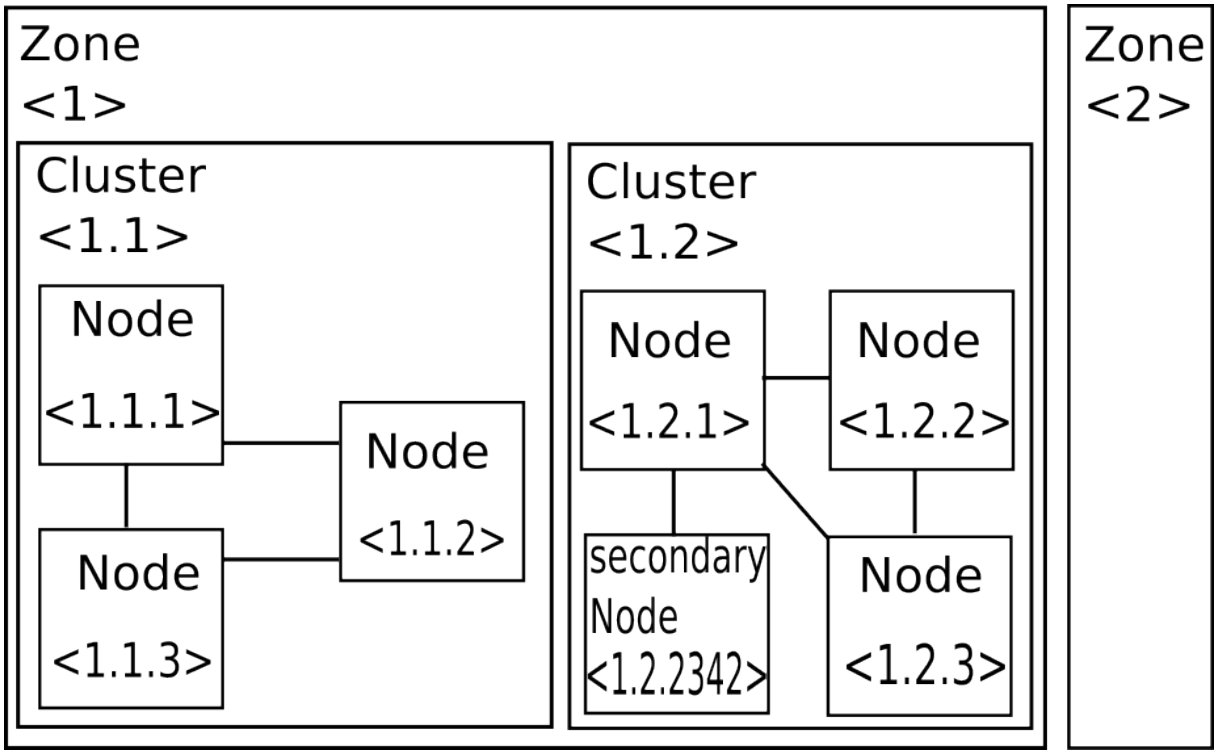
- Nachrichten benötigen gewöhnlich nur einen Hop, Transferzeiten kurz
- Nachrichten laufen mehrheitl. über Intra-Cluster Verbindungen
- geringer Paketverlust, Retransmits selten, hohe Bandbreite verfügbar
- Paket-Checksummen – wenn nötig – durch Hardware
- Anzahl an sendenden Knoten ist i.a. statisch
- Sicherheit in geschlossenen Clustern von geringer Bedeutung

TIPC: Funktionale Übersicht



TIPC-Topolgie

Adressnotation: <Zone.Cluster.Node>



TIPC: Funktionale Adressierung

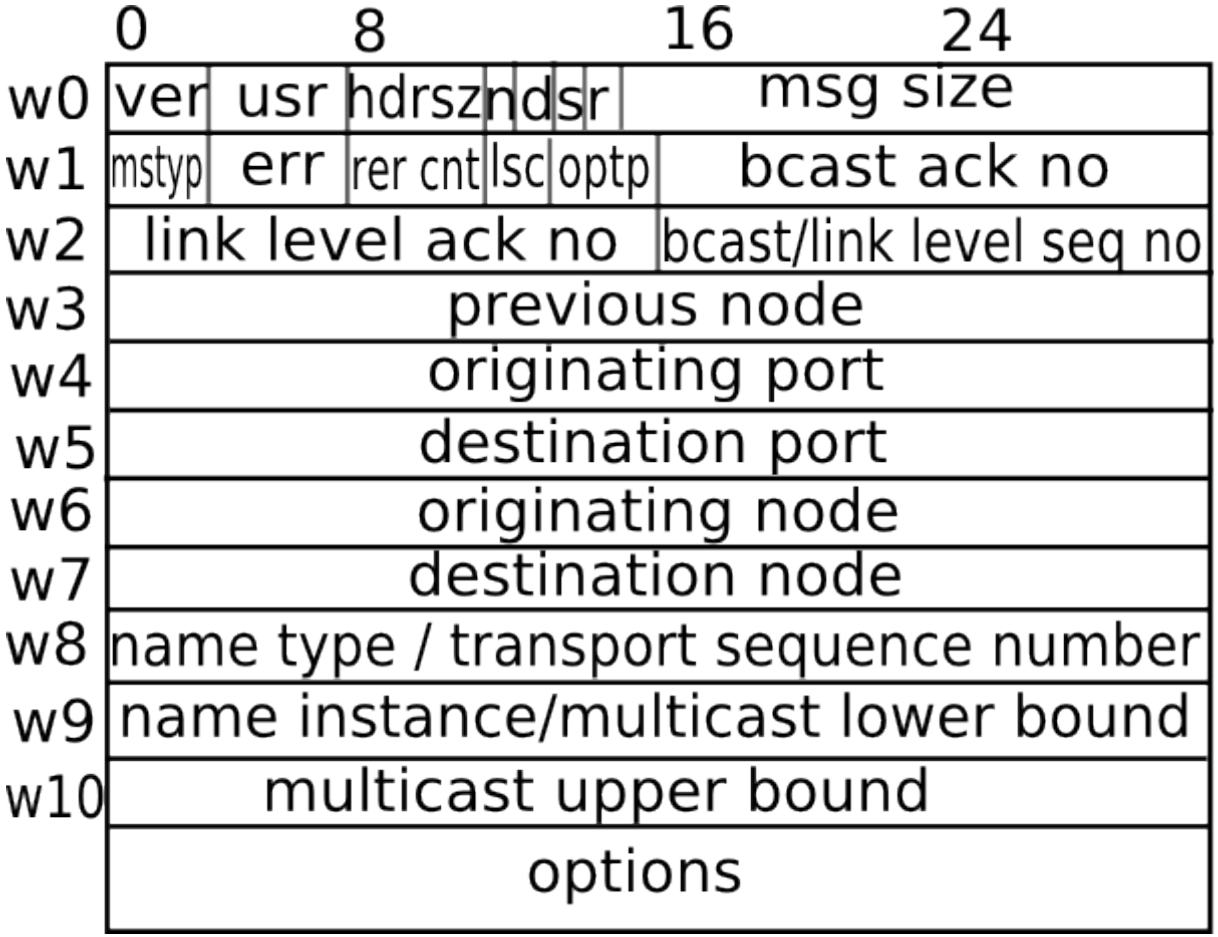
- Adressierung erfolgt über Diensttyp/Instanznummern
- . . . werden in `sockaddr_tipc` hinterlegt
- Anbieter: `socket`, `bind`, `recv`
- Sender: `socket`, `sendto`

```
int sockfd = socket(AF_TIPC, SOCK_RDM, 0);
struct sockaddr_tipc server;
server.addr.name.name.type = TYPE_BANDWITH_MGMT;
server.addr.name.name.instance = 1;
%[..]
res = bind(sockfd, (struct sockaddr *)&server, servlen);
```

Socket-Typen

- `SOCK_DGRAM` Datagramme (verbindungslos, unzuverlässig)
- `SOCK_RDM` Dategramme (verbindungslos, zuverlässig)
- `SOCK_SEQPACKET` Datagramme (verbindungsorientiert, sequentiell, zuverlässig)
- `SOCK_STREAM` Bytestrom (verbindungsorientiert, sequentiell, zuverlässig)

TIPC Payload Header



Motivation

- "35% faster than TCP" [JM04]
- Stimmt leider (so) nicht mehr:

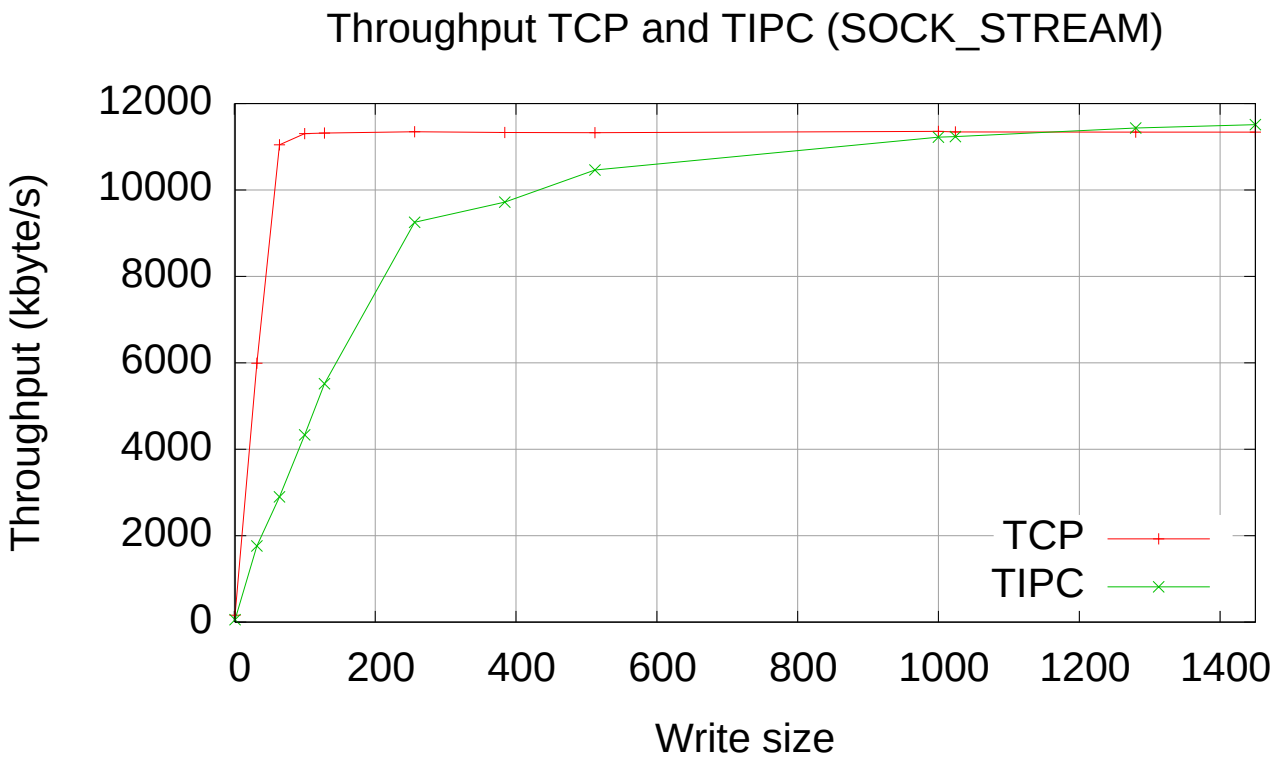
```
Performance degraded during the process of including TIPC
in the standard kernel. [..]
```

Wie messen?

1. Durchsatz
   - Daten fester größe übermitteln
   - Zeitdauer bestimmen
2. Latenz / RTT
   - jedes Datenpaket auf Applikationsebene quittieren → 'Pingpong'
   - Um Netz möglichst auszulasten: je 16 Sender/Receiver Paare, Wert pro Paket errechnen/mitteln (JM04)

Analyse: Testaufbau

2 Knoten, 100MBit-Ethernet, TIPC 1.7.2



Auswertung

- TCP: Nagle Algorithmus in Aktion: guter Durchsatz auch unter 'schlechten' Bedingungen

Probleme:

- 'echter' Flaschenhals ist das Netzwerk, nicht Software
  $\rightarrow$ kaum sinnvolle Schlussfolgerungen möglich

- besser wären G oder sogar 10G Ethernet

- Scheinbar kaum Spielraum für Verbesserungen: `copy_from_user`, Netzwerktreiber, . . . benötigen ihre Zeit

- TCP erlaubt viele Einstellungen (`NODELAY`, diverse sysctls, . . .)

Latenzen

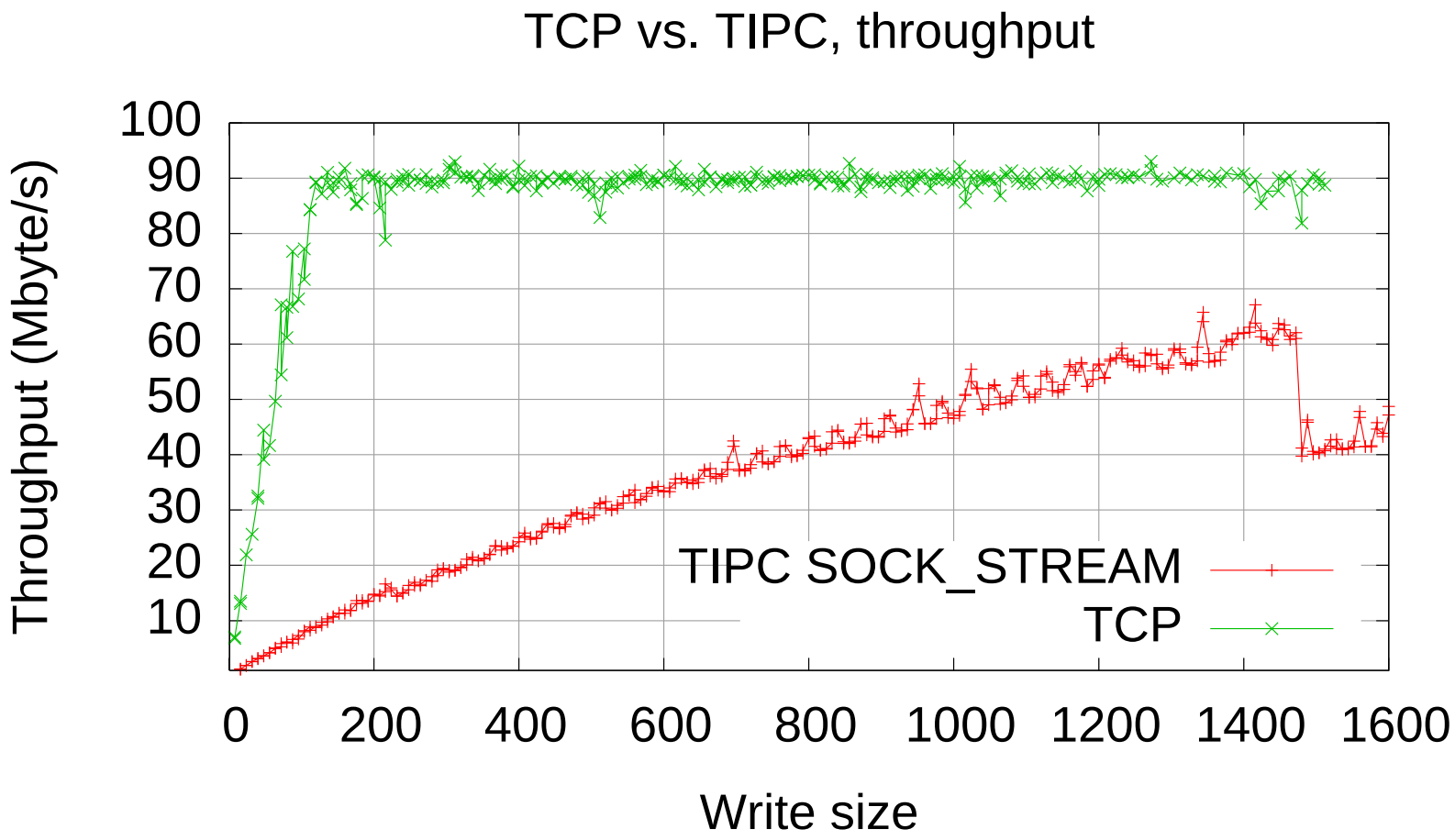
- TCP: Nagle abgeschaltet, `net.ipv4.tcp_low_latency = 1`
- (μ)-Optimierungen (`kfree_skb()`, Locking) brachten (minimale) Verbesserung

Msg size [bytes]	TCP [μs]	TIPC 1.7.2 [μs]	TIPC 1.7.3 [μs]
64	37	34	33
256	43	39	38
1024	93	90	89
4096	386	350	349

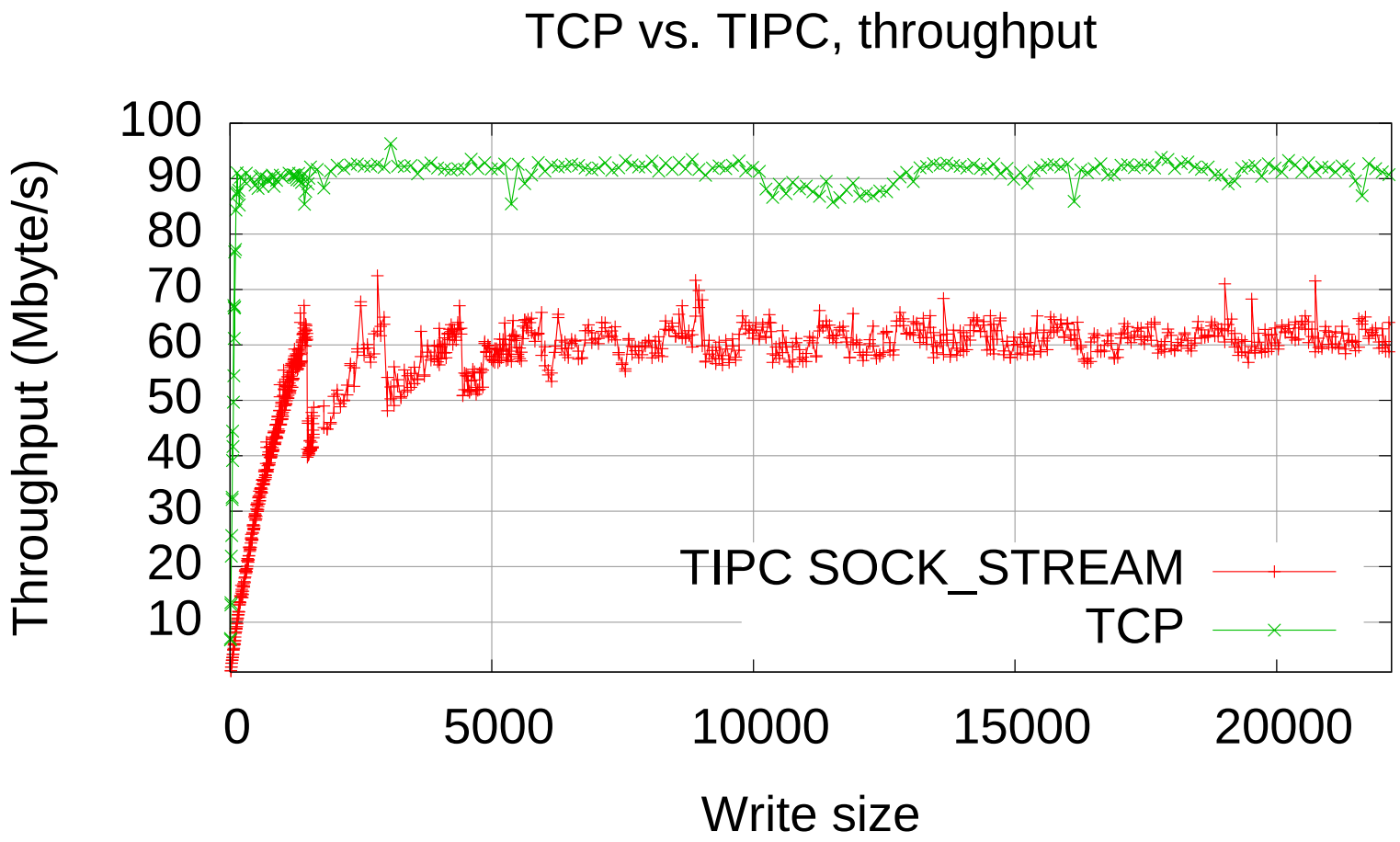
Durchsatzmessungen mit GB-Ethernet

- Sender: AMD Athlon 64 X2 (dual core) 4600+, eth: tg3
- Empfänger: Intel Xeon (2.8 GHz), eth: Intel e1000
- rx/tx checksums, s/g, TSO, etc. eingeschaltet
- Kein Switch

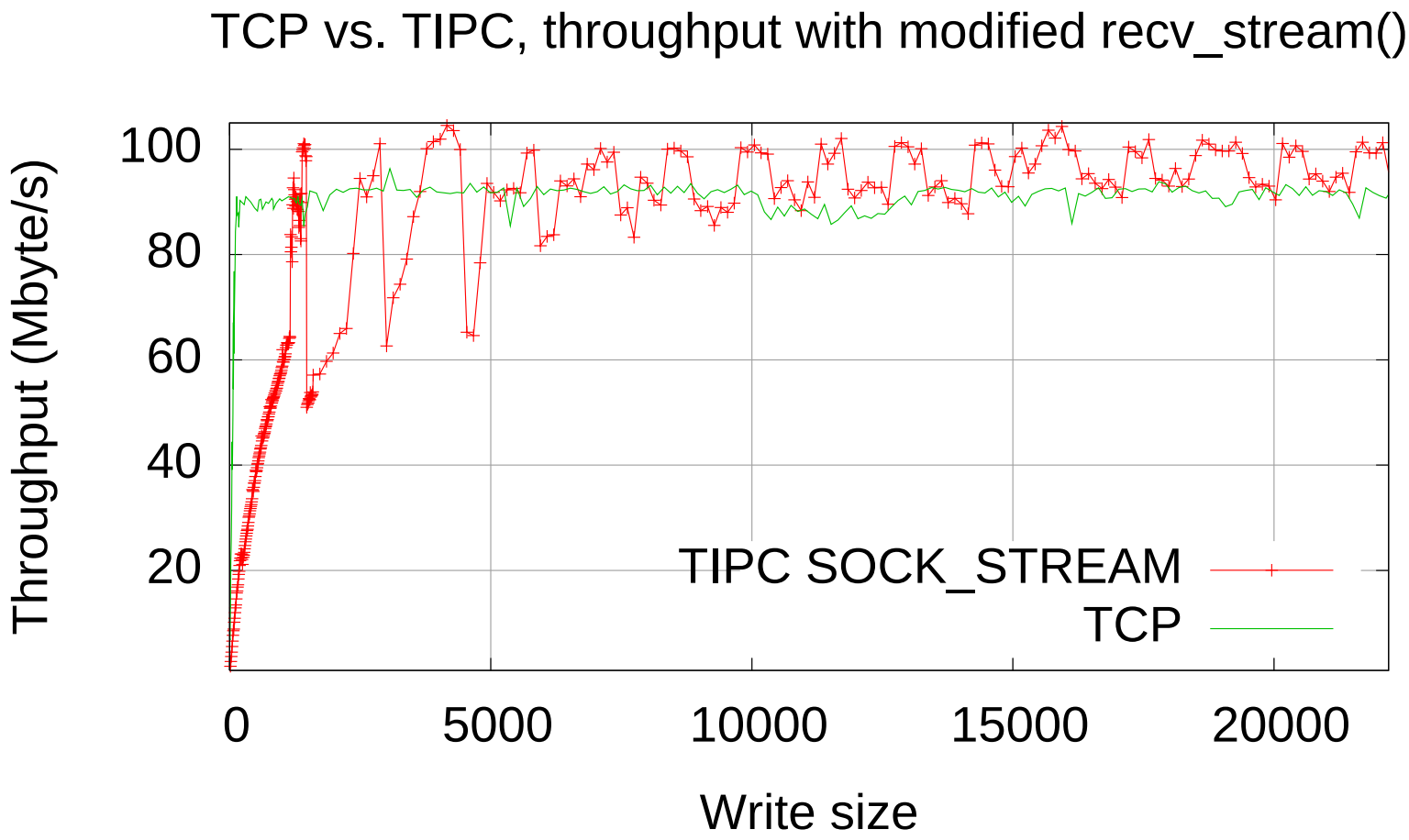
Stream Sockets, TCP vs TIPC



Stream Sockets, TCP vs TIPC



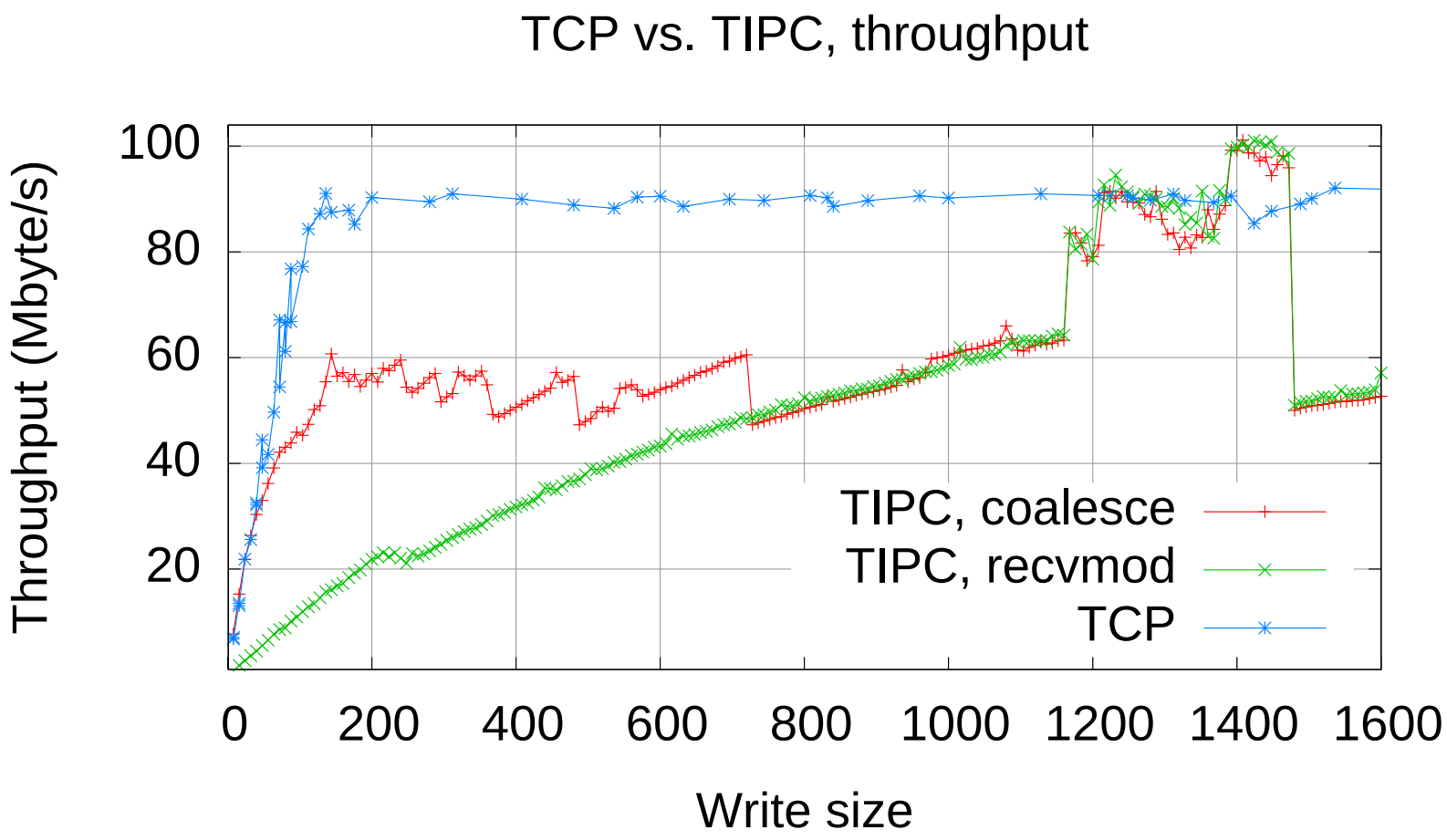
Stream Sockets, TCP vs TIPC, fix



Nagle-Algorithmus

- Vermeidung von Tinygrams
  - Anzahl Daten $\geq$ MSS
    oder: Kein ACK ausstehend $\rightarrow$ Paket senden
  - sonst: Daten (noch) nicht senden
- Transparent für Empfänger
- keine gesonderten Timer nötig

TIPC-Nagle


TCP vs. TIPC, throughput
Throughput (Mbyte/s)
100
80
60
40
20
0
200
400
600
800
1000
1200
1400
1600
Write size
TIPC, coalesce
TIPC, recvmod
TCP

Fazit

- Durchsatz und Latenz sind gut
- Latenz evtl. Verbesserungsfähig (Stichwort `skb_clone()`, `SOCK_DGRAM` nicht in Queue halten, . . .)
- Nächste logische Optimierungsschritte:
  - Speicher/Ressourcenbedarf verkleinern (Code & Daten)
  - klarere Trennung zwischen `_RDM` und `_DGRAM`
  - SMP-Skalierung & Review des Locking-Modells

Quellen

JM04 J. Maloy: "Providing Communication for Linux Clusters".
In: Proceedings of the Linux Symposium, Vol. 2; Juli 2004.

MS06 J. Maloy, A. Stephens "Transparent Inter Process Communication Protocol"
`http://tipc.sourceforge.net/doc/draft-spec-tipc-02.txt`

UTSL `vim /usr/src/linux/net/tipc/*c`

- Testprogramme:
  - Durchsatz: `http://netsend.berlios.de/`
  - Latenz: `http://www.strlen.de/tipc/tipc_pp_0.0.2.tar.bz2`