同時実行性及びパケット順序処理に着目したCYPHONICクライアントの実装と評価

後藤廉[1)], 眞玉和茂[1)], 相畑亮太[2)], 鈴木秀和[3)], 内藤克浩[2)]

[1)] 愛知工業大学大学院 経営情報科学研究科
[2)] 愛知工業大学 情報科学部 情報科学科
[3)] 名城大学 理工学部 情報工学科

2023年 電子情報通信学会 ソサイエティ大会
2023年 9月 15日 (金)

集中型処理モデルから分散型処理モデルへ

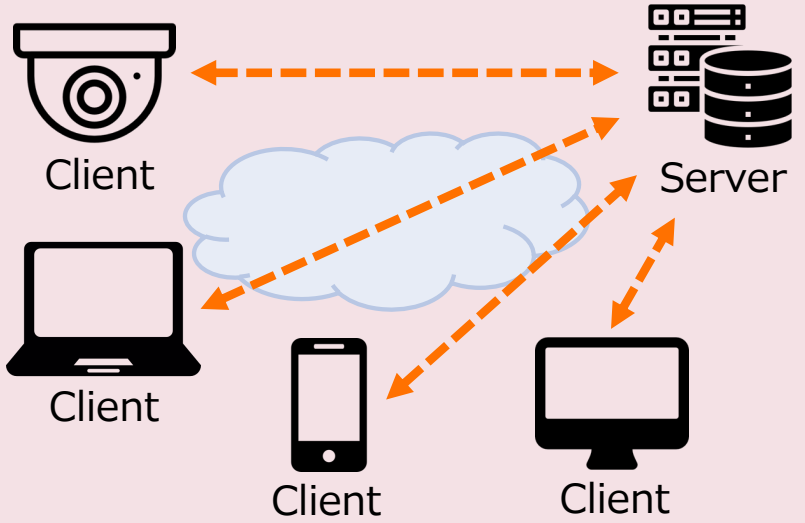
集中型処理モデル：Client-to-Server

サーバでの一極集中処理により

端末の管理や通信トラフィックを制御

➡ 通信量の増加に伴うネットワーク網や

サーバへの負荷集中・遅延の増大が懸念



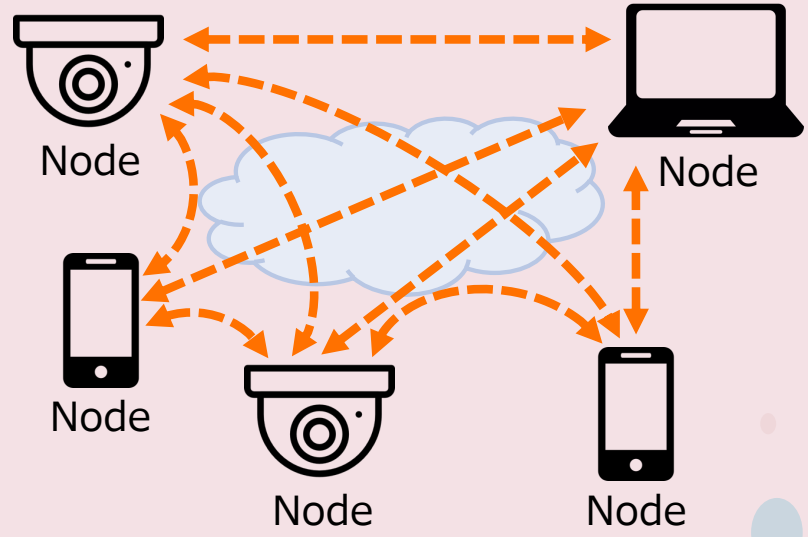
分散型処理モデル：Peer-to-Peer

一つのタスクを複数の端末に分散して

協調処理を行うことでサービスを実現

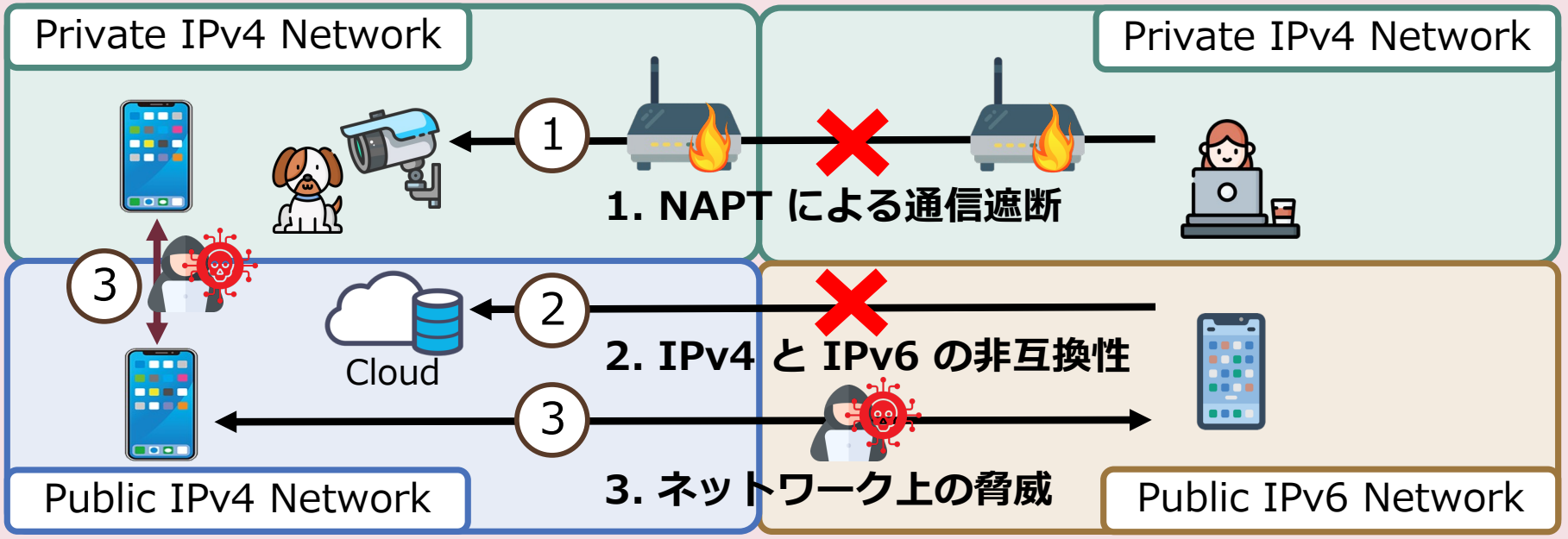
➡ クラウドサービスに対する負荷の軽減や

遅延の改善が可能



分散処理モデルの実現には Peer-to-Peer による
機器間の相互接続及び直接通信機構が必要

Peer-to-Peer におけるネットワーク課題



NAPT：Network Address Port Translation

課題1：NAPT によるインバウンドトラフィックの遮断（通信接続性）
- 内部ネットワークに存在する端末を外部ネットワークから隠蔽

課題2：IPv4 と IPv6 間の非互換性（通信接続性）
- IPv4 と IPv6 ではパケットフォーマットが異なる

課題3：相互接続・直接通信に伴うネットワーク脅威の考慮（機密性・完全性）
- 暗号化やアクセス制御技術の導入による安全な通信の実現

CYPHONIC 概要

CYber PHysical Overlay Network over Internet Communication

P2P モデルにおける課題を包括的に解決してセキュアな通信サービスを提供

NAPT 越え：ネットワーク環境に応じた経路指示に従った通信

IPv4-IPv6 通信：仮想 IP に基づいた通信により実 IP のバージョン差を隠蔽

セキュリティ：認証の導入及びオーバーレイネットワーク上での暗号化通信

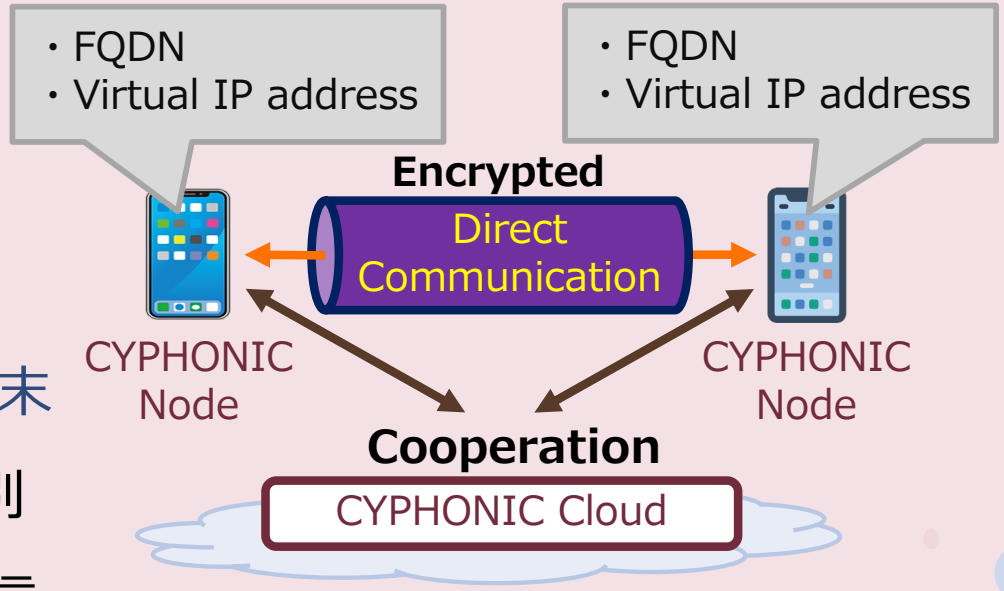
CYPHONIC クラウド

➡ 端末の認証及び管理機能を提供

CYPHONIC ノード

➡ CYPHONIC Daemon を搭載した端末

- 一意となる FQDN による端末の識別
- 不変な仮想 IP アドレスに基づく通信

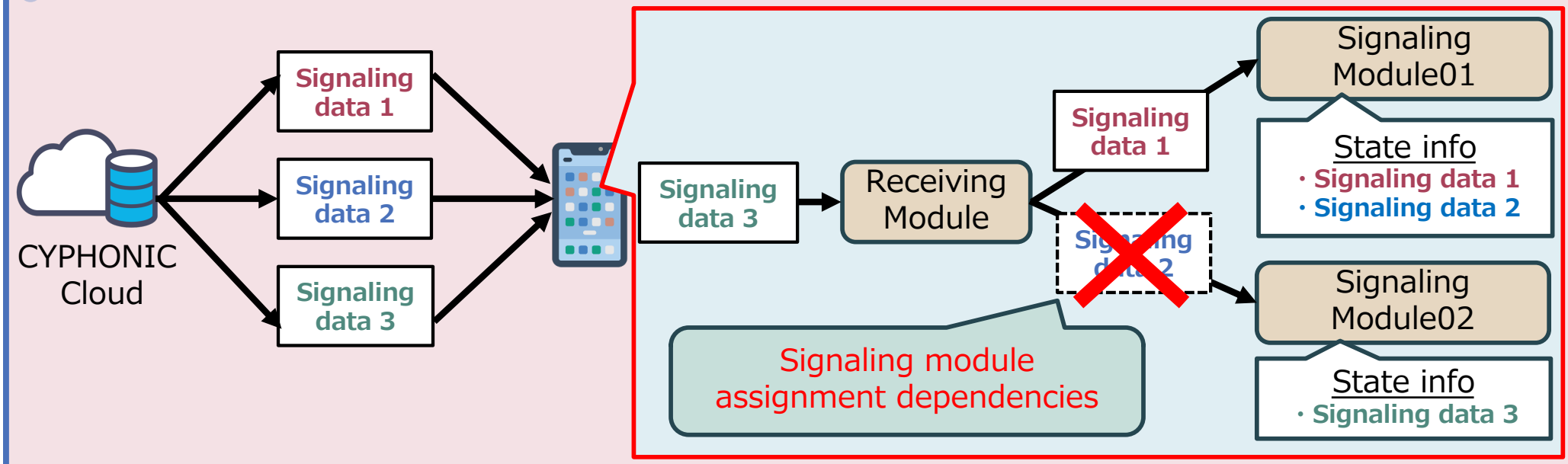


FQDN：Fully Qualified Domain Name

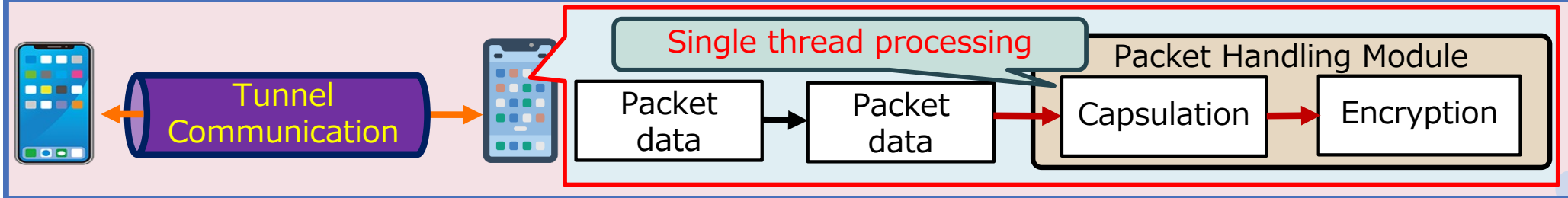
CYPHONIC ノード は CYPHONIC クラウド と連携することにより
端末間で自律的にトンネルを構築して直接通信を確立

CYPHONIC クライアント 課題

トンネル構築処理に伴うシグナリングモジュールに依存関係が存在



パケット処理モジュールは受信パケットを直列に処理



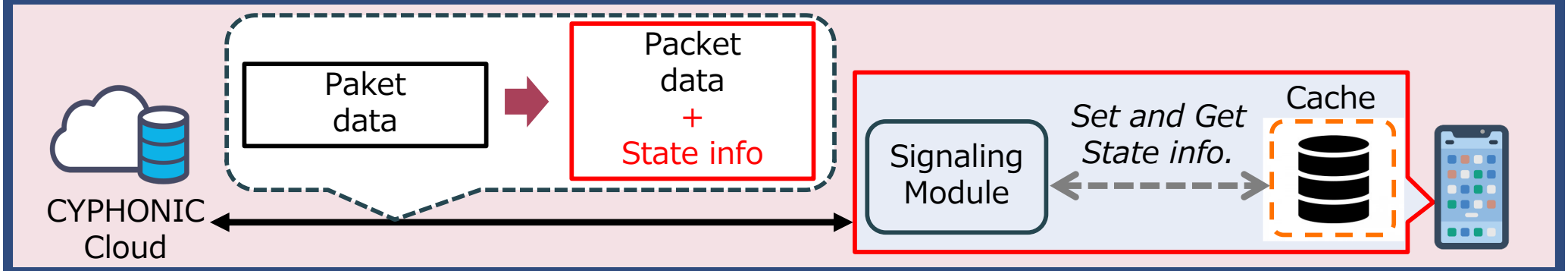
・一部の処理モジュールがステートを持つことで並行処理が困難
・単一モジュールによる直列処理によってスループットが低下

提案システム

同時実行性及びパケット順序制御に着目した
マルチスレッド非同期処理方式の提案

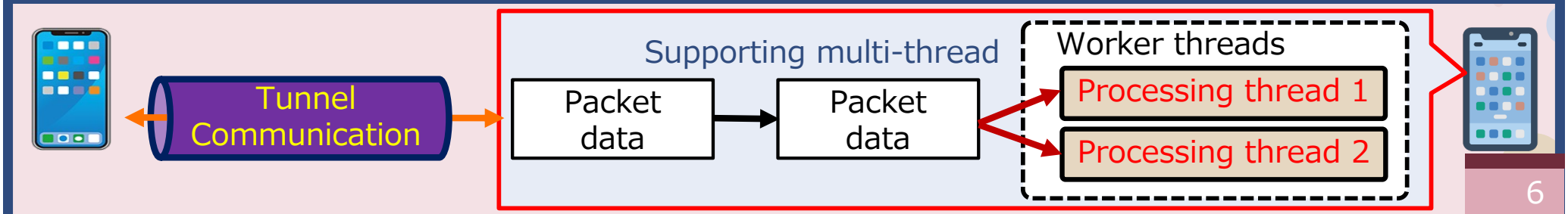
トンネル構築に伴うステート情報を内部処理から独立

- シグナリングメッセージ内部にステート情報を追加
- 情報を一時的に保持するインメモリキャッシュの追加

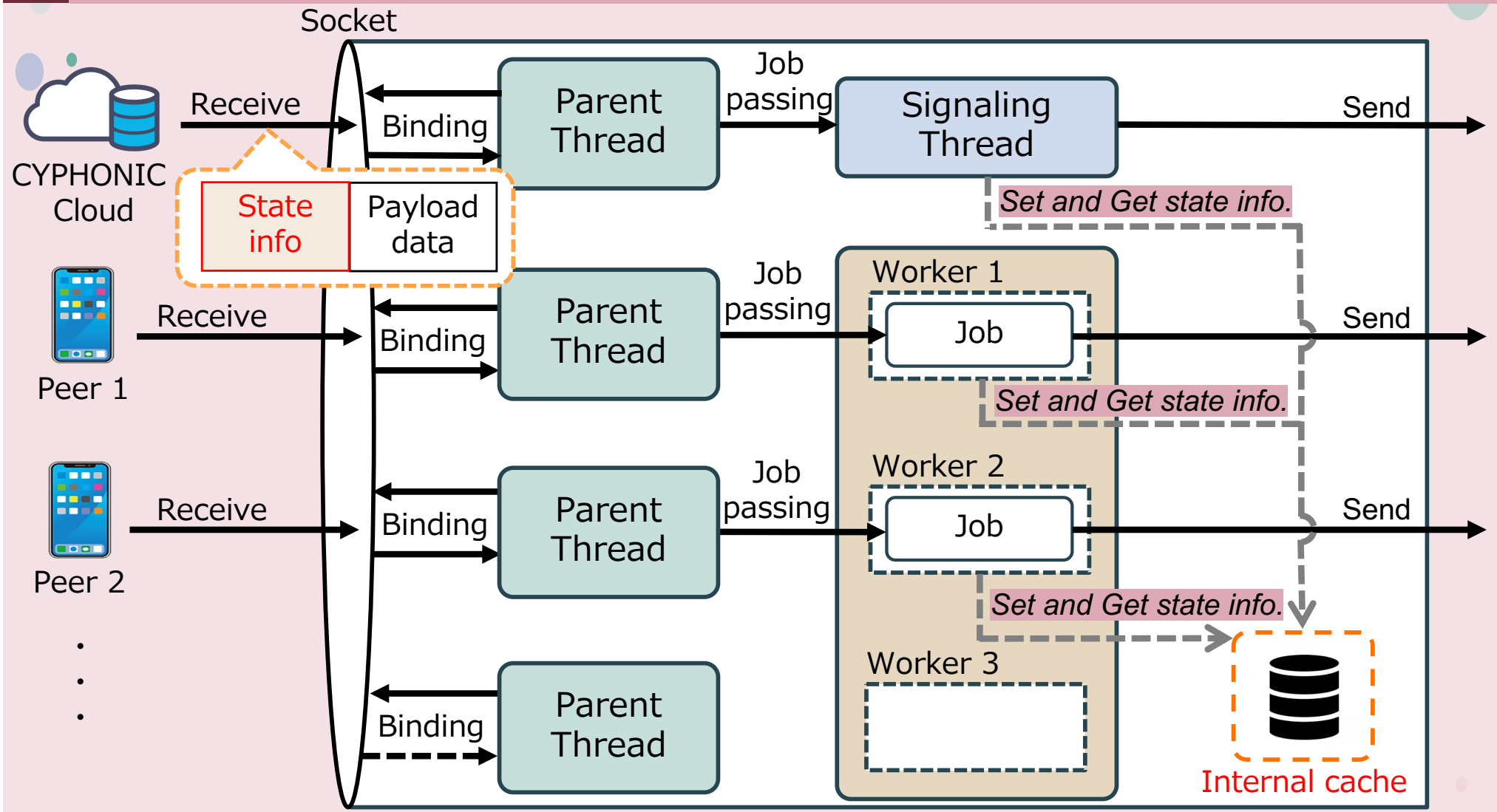


非同期実行におけるパケット逐次処理及び順序制御機構の追加

- カプセリング処理・暗号/復号処理のマルチスレッド化
- 受信時点と送信時点におけるパケット順序の維持

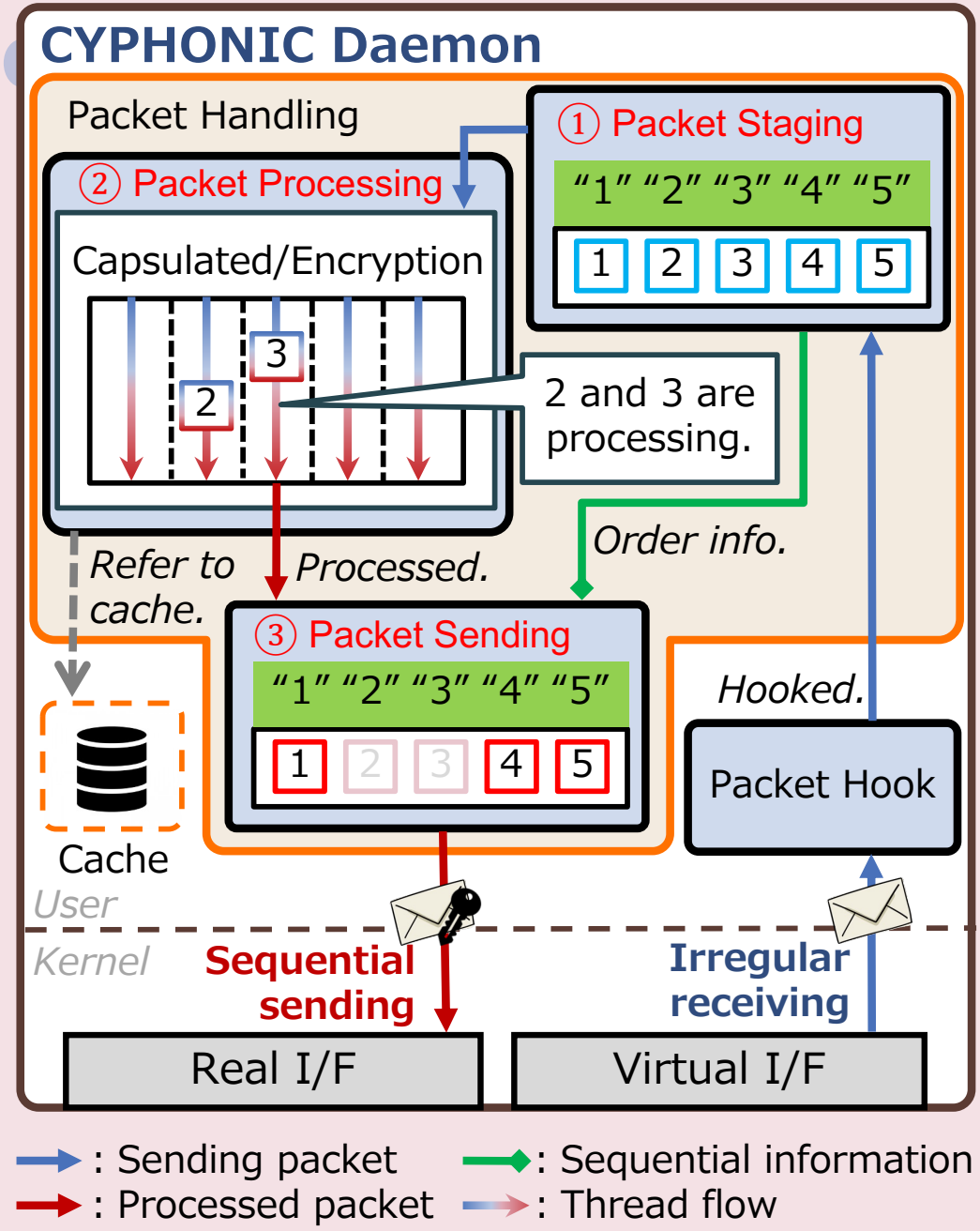


マルチスレッドによるトランザクション方式



シグナリングメッセージにステート情報を追加しつつ
内部キャッシュを用いてワーカースレッド間でデータを共有することで
クライアントプログラムのマルチスレッド対応が可能

パケット順序付け及び逐次処理モデル



① Packet Staging Module

受信パケットをバッファに格納し

受信順序を保存

② Packet Processing Module

処理をワーカースレッドに割り当て

カプセル化・暗号化を非同期実行

③ Packet Sending Module

処理済みパケットをキューに

追加すると共に受信順序に基づき送信

非同期実行及びワーカースレッドの
処理状況に依らず送受信において
パケットの順序維持が可能

検証項目及び評価環境

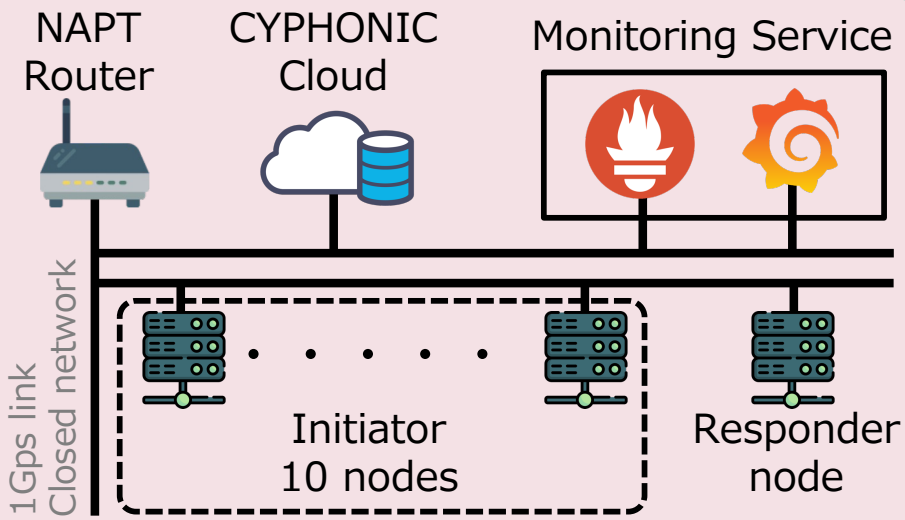
複数トンネル確立時における CYPHONIC ノードの性能評価

ネットワーク性能

- TCP/UDP スループットの計測
  - ➡ iperf3 を使用
- ICMP による通信遅延の計測
  - ➡ ping を使用

内部処理動向

- OS スレッド数 と Goroutine 数の動向
  - ➡ NodeExporter / GoMetrics を使用



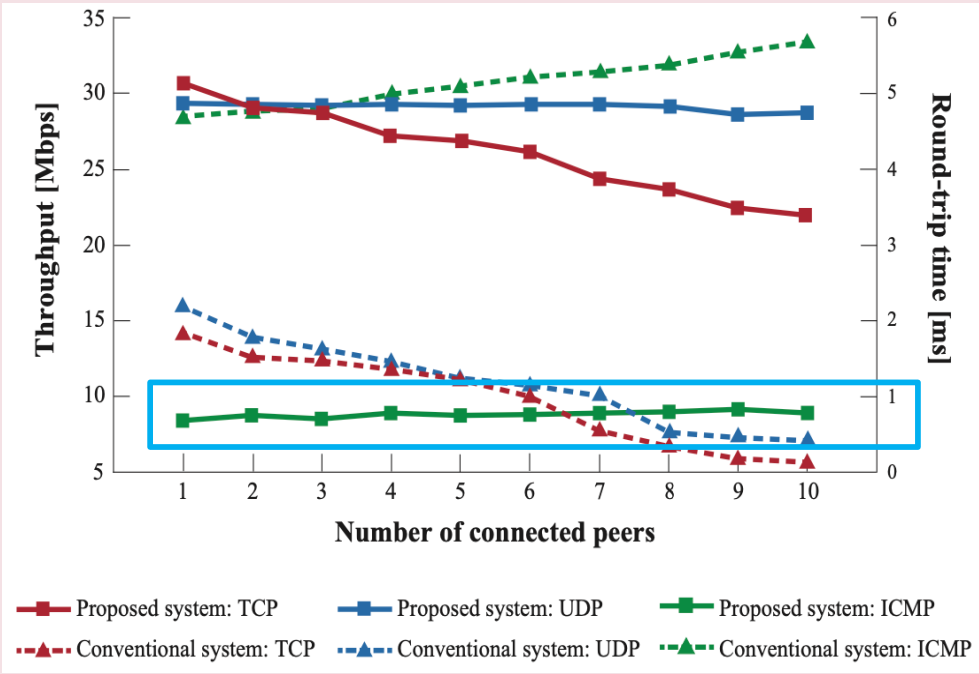
Virtual Machine (CYPHONIC Node)	
OS	Ubuntu 22.04 Jammy Jellyfish
CPU	Intel(R) Core(TM) i9-13900 CPU@5.60GHz, 2-cores / 2-threads
Memory	1 GiB

Implementation	
Runtime	Go version 1.20
Worker thread	Goroutines

閉域ネットワーク網 に 1 台の通信待受端末と 10 台の通信開始端末を配置

➡ 最大 10 台のピアノードとトンネルコネクションを確立して通信

提案処理モデル 評価結果



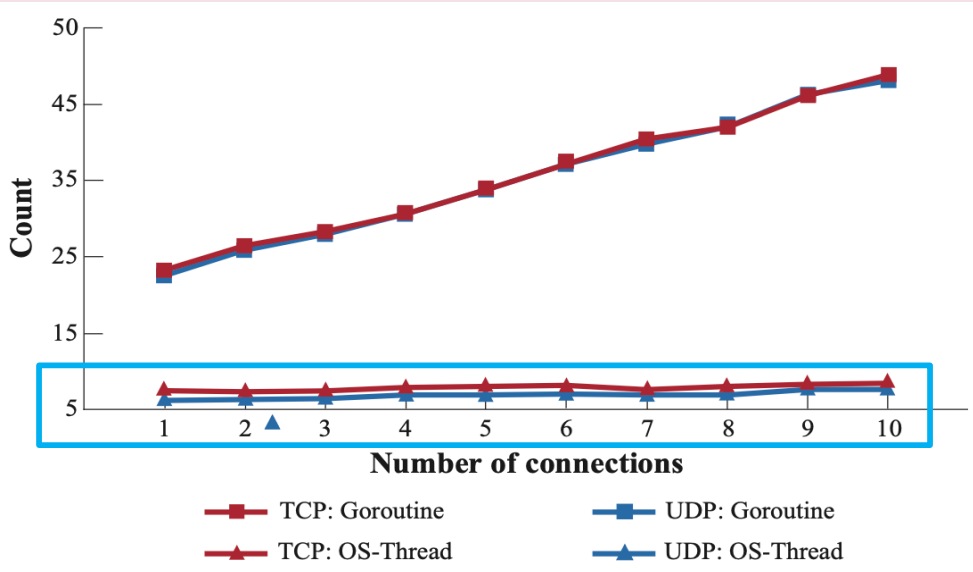
単一コネクションの通信に着目して

- TCP：約 16.9 Mbits/sec 向上
- UDP：約 13.1 Mbits/sec 向上
- RTT：約 4.0 ms 改善

従来の処理方式ではコネクションの増加に伴う RTT が増加する傾向

➡ 提案方式では一定に保つことが可能

RTT：Round-Trip Time



コネクション数の増加に伴い

- ワーカースレッド数：増加
- OS スレッド数：一定

スレッド生成やコンテキストスイッチに伴うオーバーヘッドを OS から隠蔽して最小限に抑制

まとめ

CYPHONIC クライアントプログラムにおける
同時実行性及びパケット順序制御に着目した
マルチスレッド非同期処理方式の提案

トンネル構築に伴うステート情報を内部処理から独立
- シグナリングメッセージ内部にステート情報を追加
- インメモリキャッシュストアによる通信データの管理

非同期実行におけるパケット逐次処理及び順序制御機構の追加
- カプセリング処理・暗号/復号処理のマルチスレッド化
- 受信時点と送信時点におけるパケット順序の維持

スループットの大幅な向上とコネクション増加に伴う
通信遅延を一定に維持可能

近年の端末処理性能が向上していることからも
本モデルの採用によって通信性能の更なる改善が期待

以下、予備スライド

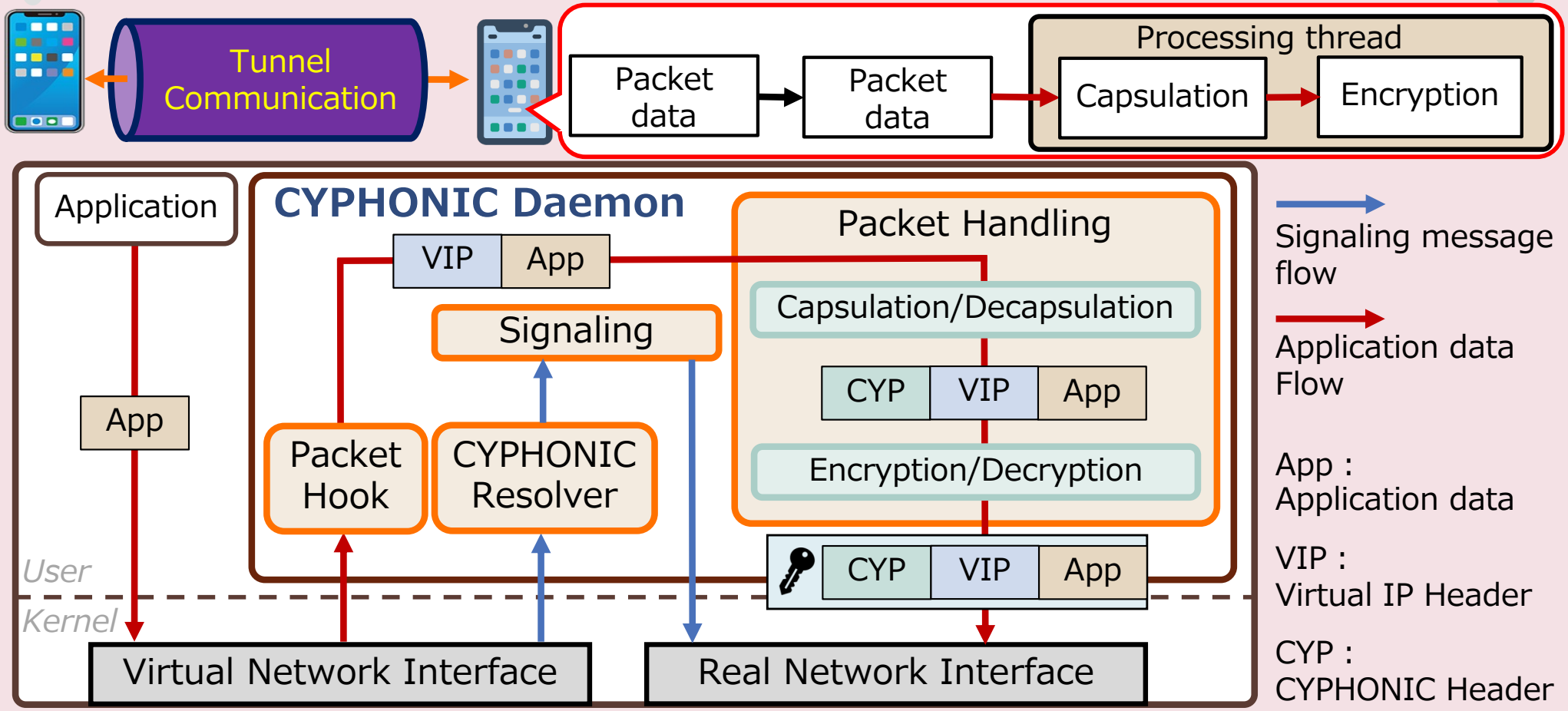
既存技術の課題と CYPHONIC の位置付け

	NAT Traversal optimization	IPv4 - IPv6 Dual stack	Network Security	Mobility Transparency
STUN	○	×	×	×
ICE	○	×	○	×
SoftEther	○	×	○	×
Tailscale	○	×	○	×
DSMIPv6	×	○	×	○
QUIC	△	○	○	○
CYPHONIC	○	○	○	○

端末間の継続的な相互説接及び直接通信を実現する包括的な技術が必要

CYPHONIC はオーバーレイネットワークを用いて
セキュアなエンド間通信を実現する通信フレームワーク

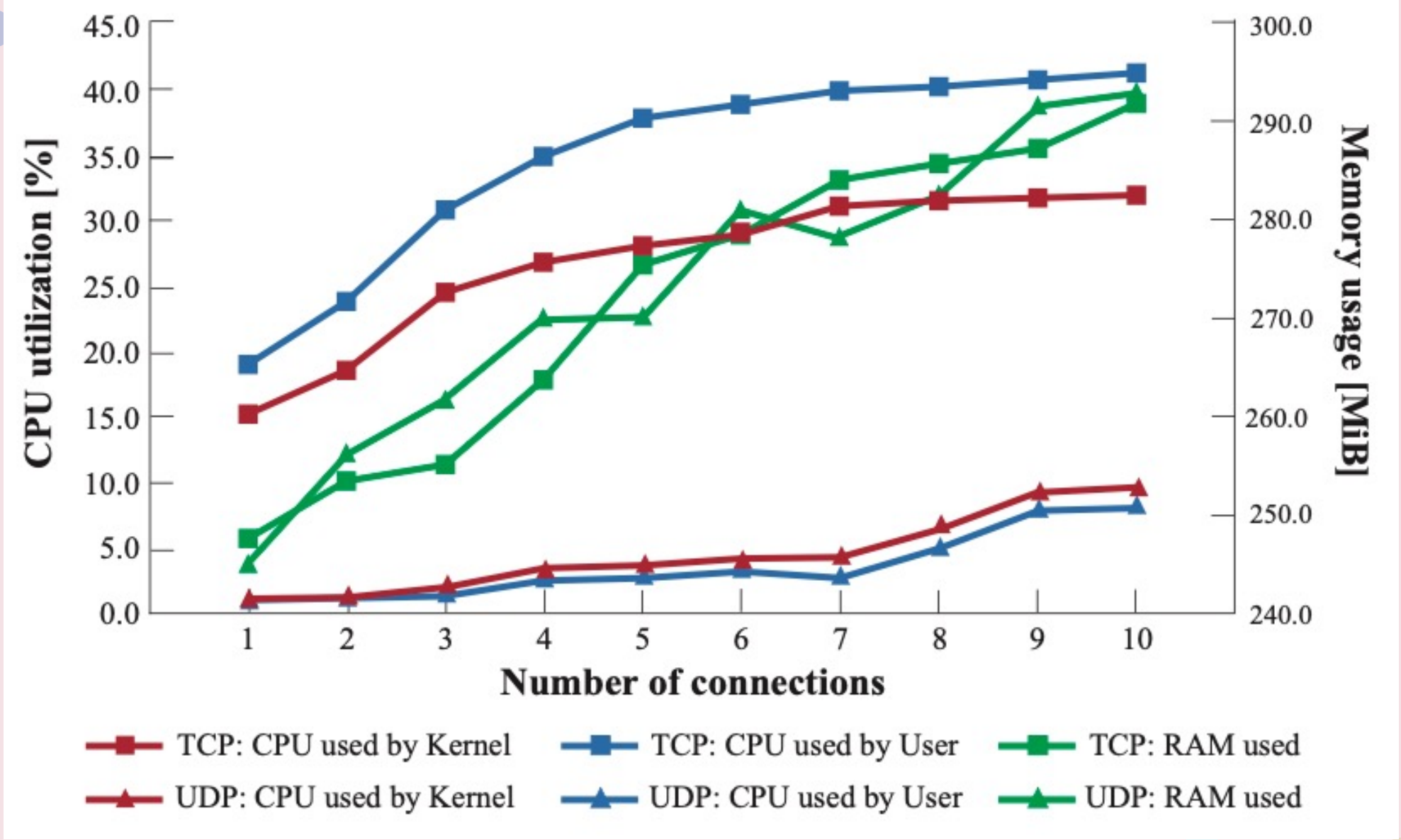
CYPHONIC クライアントシステム



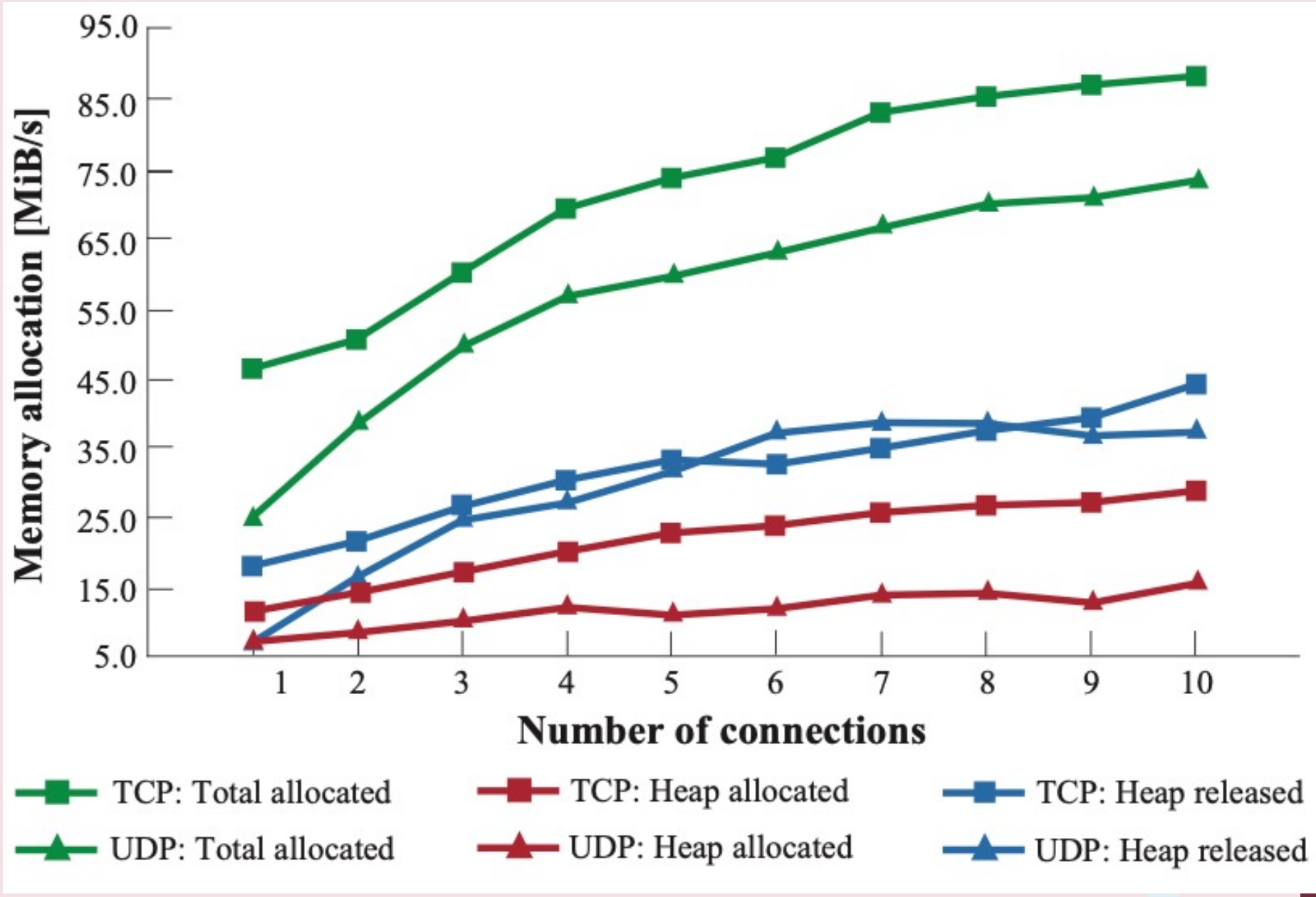
従来の CYPHONIC Daemon はシングルスレッドによる
シンプルなパケット処理モデルを実装

単一モジュールの処理負荷が別で動作するモジュールに影響することで
スループットの低下や遅延の増大が発生

リソース使用量の動向


CPU utilization [%]
45.0
40.0
35.0
30.0
25.0
20.0
15.0
10.0
5.0
0.0
Memory usage [MiB]
300.0
290.0
280.0
270.0
260.0
250.0
240.0
1
2
3
4
5
6
7
8
9
10
Number of connections
TCP: CPU used by Kernel
TCP: CPU used by User
TCP: RAM used
UDP: CPU used by Kernel
UDP: CPU used by User
UDP: RAM used

APM メトリクスの動向



検証・試験の目的及びコンセプト

目的１：スループット検証

コネクション確立に伴う処理状況の調査

目的２：ヒート試験

CYPHONIC Daemon 長時間稼働時における安定稼働性

コネクション数の調整

コネクション辺り に要する試験時間

- 15分(試験) + 5分(準備) x 3回 x 100回 ({UP: 1->10 / DOWN: 10->} x 5) iteration = 100h

全プロトコルにおける検証時間（所用日数）

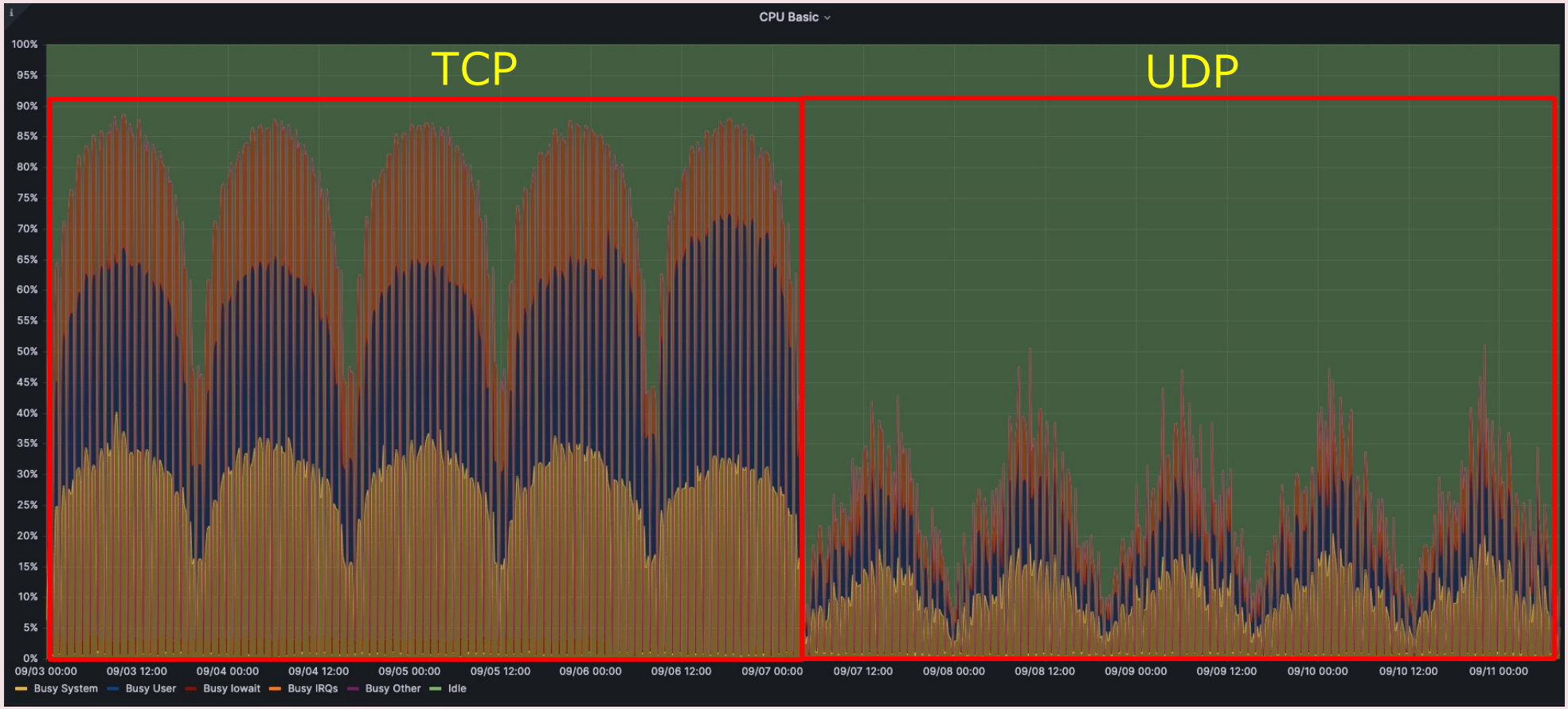
- 100h x 3 プロトコル（TCP, UDP, ICMP）= 約12.5日

➡ 一度試験を回し始めると数時間起動し続ける

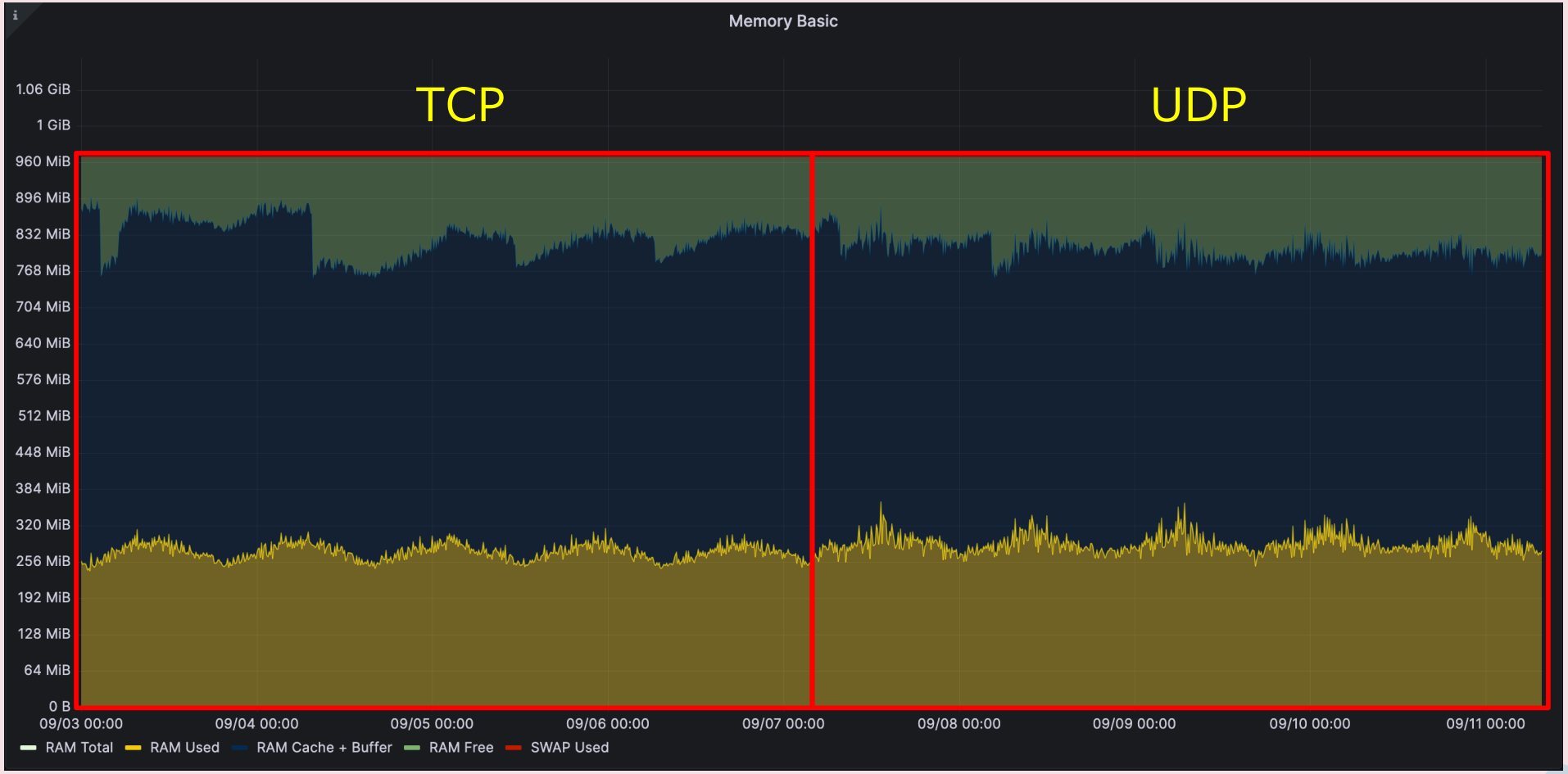
双方の試験を行うため、コネクション数を調整して検証を実施

台数を 3, 5, 10, ... と順に増加させることで基礎評価を取得

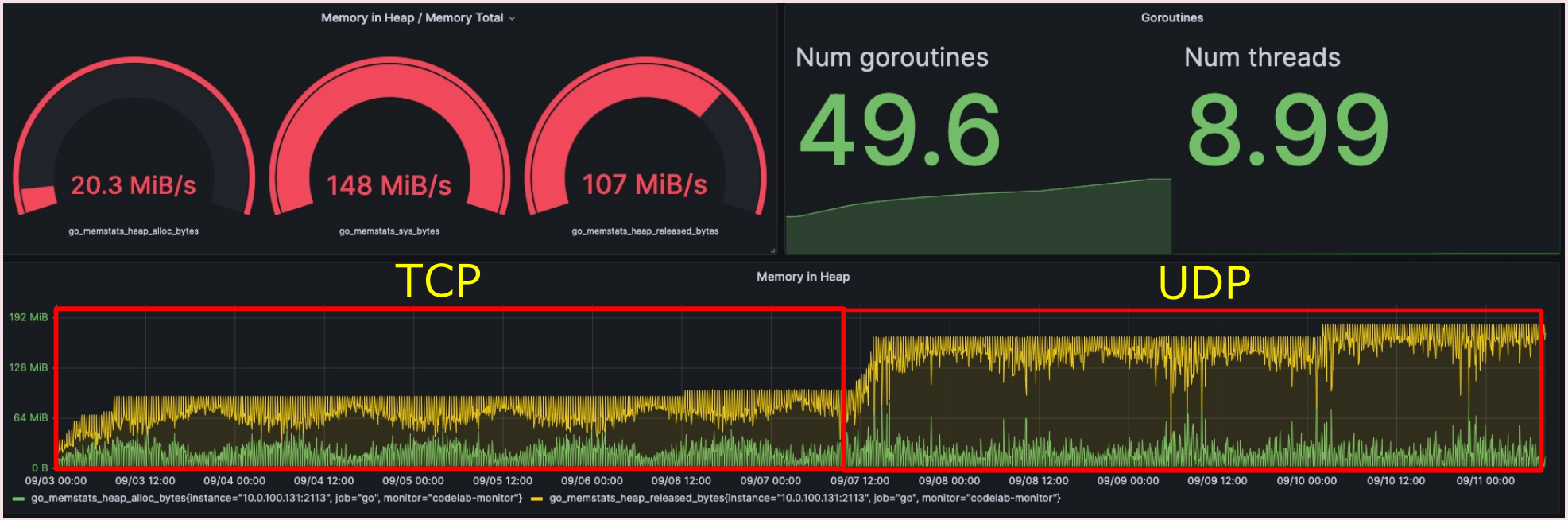
コネクション増減試験におけるCPU動向



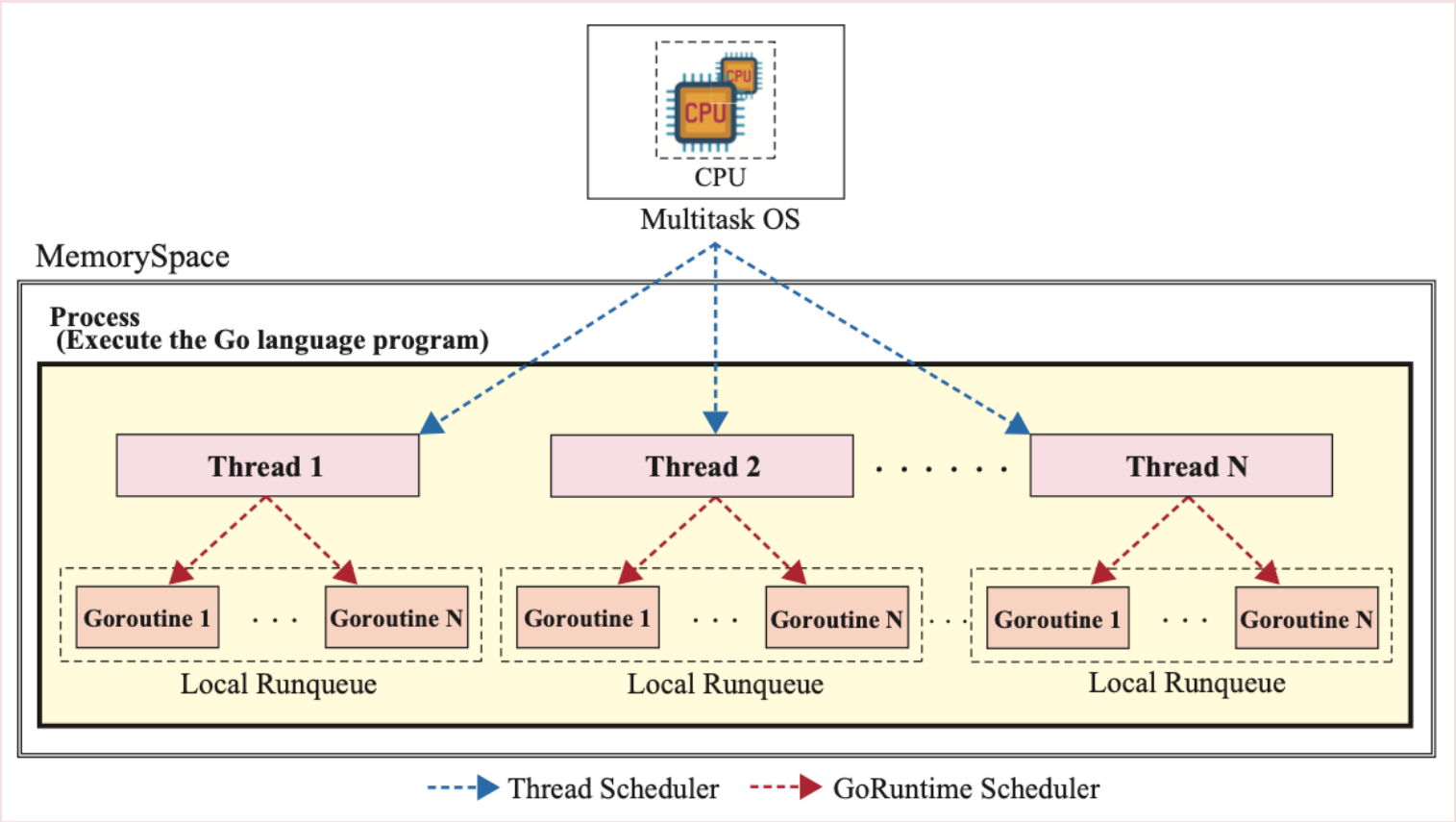
コネクション増減試験におけるメモリ動向



コネクション増減試験におけるAPM動向



Goroutine の処理モデル



Hyper-threading Kernel（Threads）

- *M* 個 の物理コアに対して同時に *N* 個 の処理が可能な *M:N* スケジューラ

GoRuntime（Goroutines）

- *M* 個 の論理コアに対して同時に *N* 個 の処理が可能な *M:N* スケジューラ